Simulazioni montecarlo di rivelatori di radiazioni con Geant4

Davide Valsecchi 785396

16 gennaio 2018

Indice

1 Esercizio 1 **1**
- 1.1 Descrizione generale . . . 1
- 1.2 Componenti del programma . . . 1
  - 1.2.1 Main . . . 1
  - 1.2.2 Detector Construction . . . 3
  - 1.2.3 Physics List . . . 8
  - 1.2.4 PrimaryGeneratorAction . . . 9
- 1.3 Macro . . . 11
- 1.4 Risultato . . . 11

2 Esercizio 2 **13**
- 2.1 Descrizione generale . . . 13
- 2.2 Componenti del programma . . . 13
  - 2.2.1 DetectorConstruction . . . 13
  - 2.2.2 DetectorMessenger . . . 14
  - 2.2.3 SensitiveDetector . . . 15
  - 2.2.4 RunAction e EventAction . . . 16
  - 2.2.5 SiDigitizer . . . 16
  - 2.2.6 RootSaver . . . 17
- 2.3 Risultato . . . 17
  - 2.3.1 Telescopio semplice (senza DUT) . . . 17
  - 2.3.2 Telescopio con DUT . . . 20

3 Esercizio 3a **23**
- 3.1 Descrizione generale . . . 23
- 3.2 Componenti del programma . . . 23
  - 3.2.1 Analysis . . . 23
  - 3.2.2 StackingAction . . . 24
  - 3.2.3 SteppingAction . . . 25
- 3.3 Risultati . . . 26
  - 3.3.1 Pioni . . . 26
  - 3.3.2 Elettroni . . . 28

4 Esercizio 3b **32**
4.1 Descrizione generale . . . 32
4.2 Componenti del programma . . . 32
4.2.1 DetectorConstruction . . . 32
4.2.2 PhysicsList . . . 33
4.2.3 Analysis . . . 34
4.3 Risultati . . . 35

5 Esercizio 4 **38**
5.1 Descrizione generale . . . 38
5.2 Esercizio 4a . . . 38
5.3 Esercizio 4b . . . 41
5.4 Esercizio 4c . . . 44

6 Esercizio 6 **49**
6.1 Descrizione generale . . . 49
6.2 Componenti del programma . . . 49
6.2.1 DetectorConstruction . . . 49
6.2.2 DetectorMessenger . . . 50
6.2.3 SensitiveDetector . . . 51
6.2.4 Salvataggio dati . . . 52
6.2.5 PhysicsList . . . 53
6.3 Risultati . . . 53
6.3.1 Catodo in polietilene . . . 53
6.3.2 Confronto materiali . . . 54

A Compilazione con CMake **60**

Capitolo 1

Esercizio 1

1.1 Descrizione generale

Lo scopo di questo esercizio è simulare un rivelatore composto da tre parti:

- un telescopio costituito da tre piani con 600 strip di silicio
- un calorimetro elettromagnetico formato da cristalli di tungstato di piombo
- un calorimentro adronico costruito con 80 strati di ferro alternati a strati di argon liquido come elemento sensibile.

1.2 Componenti del programma

1.2.1 Main

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. Si instanziano poi le classi create dall'utente per gestire la costruzione del detector (`DetectorContruction`), i processi fisici attivi (`PhysicsList`) e la generazione delle particelle primarie (`PrimaryGeneratorAction`) e si passano questi oggetti al `RunManager`.

```
#include "G4RunManager.hh"
#include "G4UImanager.hh"
#include "G4Version.hh"

#include "G4VisExecutive.hh"
#if  G4VERSION_NUMBER>=930
#include "G4UIExecutive.hh"
#else
#include "G4UIterminal.hh"
#include "G4UItcsh.hh"
```

```
#endif
#include "DetectorConstruction.hh"
#include "PrimaryGeneratorAction.hh"
#include "PhysicsList.hh"

int main(int argc,char** argv)
{
// Run manager
G4RunManager * runManager = new G4RunManager();

// Costruzione del detector
G4VUserDetectorConstruction* detector = new
    DetectorConstruction();
runManager->SetUserInitialization(detector);

// Impostazione delle fisica
G4VUserPhysicsList* physics = new PhysicsList();
runManager->SetUserInitialization(physics);

// Impostazione della sorgente di eventi primari
G4VUserPrimaryGeneratorAction* gen_action = new
    PrimaryGeneratorAction();
runManager->SetUserAction(gen_action);

[...]
```

A questo punto si inizializza il kernel di Geant4 e il gestore della visualizzazione `G4VisManager`. Infine è necessario creare l'interfaccia grafica per l'utente con l'`G4UIManager`. Il controllo della simulazione avviene attraverso una macro o comandi inseriti direttamente dall'utente nell'interfaccia interattiva.

```
// Initialize G4 kernel
runManager->Initialize();

//Initilize the visualization manager
G4VisManager* visManager = new G4VisExecutive();
visManager->Initialize();

// Get the pointer to the User Interface manager
G4UImanager * UImanager = G4UImanager::GetUIpointer();

if (argc!=1) {  // batch mode
  // in questo caso viene eseguita una macro da un file
  // senza aprire l'interfaccia utente
  G4String command = "/control/execute ";
  G4String fileName = argv[1];
  UImanager->ApplyCommand(command+fileName);
}
```

```
18 else {
19   // modalita' interattiva
20   #if  G4VERSION_NUMBER>=930
21   G4UIExecutive * ui = new G4UIExecutive(argc,argv);
22   //If UI has graphics execute special macro: opens OpenGL Qt
       driver
23   if (ui->IsGUI())
24     UImanager->ApplyCommand("/control/execute visQt.mac");
25   else
26     UImanager->ApplyCommand("/control/execute vis.mac");
27   #endif
28   // Si apre l'interfaccia grafica interattiva
29   ui->SessionStart();
30   delete ui;
31 }
32
33 // Alla fine del programma e' sufficiente chiamare il
     distruttore
34 // del runManager per distruggere tutti gli oggetti.
35 delete runManager;
36
37 return 0;
38 }
```

1.2.2 Detector Construction

La classe `DetectorConstruction` viene utilizzata da Geant4 per costruire il rivelatore da simulare e deve essere fornita obbligatoriamente dall'utente. In questa classe vengono definiti i materiali da utilizzare, le forme dei vari componenti del rivelatore e il loro posizionamento nello spazio. Inoltre, in questo modulo devono essere definiti gli elementi attivi del rivelatore, i cosidetti Sensitive Detectors.

La classe deriva da `G4VUserDetectorConstruction`, quindi l'utente deve fornire costruttore, distruttore e il metodo `Construct()` che deve restituire un puntatore a un `G4VPhysicalVolume` , il volume contenitore di tutta la simulazione o `World Volume`. Le varie operazioni di inizializzazione possono essere organizzate liberamente dall'utente in diversi metodi chiamati all'interno di `Construct`.

DefineMaterials Nella funzione `DefineMaterials` vengono definiti i materiali necessari per costruire il rivelatore utilizzando il database di materiali NIST. In particolare si utilizzano il silicio per il telescopio, il tungastato di piombo per il calorimetro elettromagnetico, ferro e argon liquido per il calorimetro adronico. Queste variabili sono membri privati di `DetectorConstructor` definiti nell'header.

```
1 //***************  DetectorConstrucion.cc   ************
2
```

```
void DetectorConstruction::DefineMaterials()
{
//Get Materials from NIST database
G4NistManager* man = G4NistManager::Instance();
man->SetVerbose(0);

// define NIST materials
vacuum  = man->FindOrBuildMaterial("G4_Galactic");

//Tracker
air     = man->FindOrBuildMaterial("G4_AIR");
silicon = man->FindOrBuildMaterial("G4_Si");
//Em calo
pbw04   = man->FindOrBuildMaterial("G4_PbWO4");
//Had calo
lar = man->FindOrBuildMaterial("G4_lAr");
fe = man->FindOrBuildMaterial("G4_Fe");
}
```

ComputeParameters Nella funzione `ComputeParameters` vengono preparati i parametri necessari quali le dimensioni e le posizioni delle diverse componenti. Il mondo viene impostato a una larghezza di 10 m e i rivelatori sono allineati lungo l'asse Z. I piani del telescopio sono formati da 600 strip verticali spesse 300 μm e larghe 80 μm e vengono posizionati distanziati l'uno dall'altro di 90 mm prima del calorimetro elettromagnetico. Quest'ultimo è formato da due cristalli a forma di parallelepipedo uno dentro l'altro. Infine il calorimetro adronico è costituito da 80 strati di ferro spessi 20 mm alternati a strati di argon liquido spessi 8 mm.

Construct Questa funzione viene chiamata da Geant4 per costruire il detector e deve essere definita obbligatoriamente nella classe dall'utente. All'interno di questo metodo viene inizializzato il `PhysicalVolume` che contiene tutti gli oggetti della simulazione, detto *World Volume*.

```
//***************  DetectorConstrucion.cc   ************

G4VPhysicalVolume* DetectorConstruction::Construct()
{
// Si imposta la larghezza massima del mondo.
G4GeometryManager::GetInstance()->SetWorldMaximumExtent(2.*
    halfWorldLength);
// Si chiede a Geant4 la lunghezza di tolleranza della
    simulazione
// che dipende dalla lunghezza massima del mondo
G4cout << "Computed tolerance = "
<< G4GeometryTolerance::GetInstance()->GetSurfaceTolerance()/
    mm
```

```
11 << " mm" << G4endl;
12
13 //Il solido del mondo, un cubo, viene riempito di aria.
14 G4Box * solidWorld= new G4Box("world",halfWorldLength,
       halfWorldLength,halfWorldLength);
15 // Viene creato il logicalVolume del mondo.
16 logicWorld= new G4LogicalVolume( solidWorld, air, "World", 0,
       0, 0);
17
18 // Il logical volume viene infito posizionato all'origine
       degli assi
19 G4VPhysicalVolume * physiWorld = new G4PVPlacement(0,
20     G4ThreeVector(), // at (0,0,0) // no rotation
21     logicWorld,      // its logical volume
22     "World",         // its name
23     0,          // il volume mondo ha volume madre 0
24     false,           // no boolean operations
25     0);              // il volume mondo ha copy number 0
26
27 //Ora si possono costruire i diversi detector utilizzando
       vari metodi:
28 ConstructTelescope();
29 ConstructEMCalo();
30 COnstructHadCalo();
31
32 // Il metodo Construct deve sempre restituire il
       PhysicalVolume del mondo
33 return physiWorld;
34 }
```

In generale all'interno dei Geant4 la definizione di una parte del detector è costituita da tre parti.

G4VSolid Definisce la forma geometrica e le dimensioni dell'oggetto

G4LogicalVolume Definisce il materiale e le proprietà dell'oggetto. Ad esempio a un `LogicalVolume` si può collegare un `SensitiveDetector` per registrare le interazioni con le particelle della simulazione. Inoltre in Geant4 un volume può contere altri volumi ed è il `LogicalVolume` che memorizza questa relazione.

G4VPhysicalVolume Questo oggetto costituisce la rappresentazione fisica di un `LogicalVolume` in una certa posizione nello spazio. Un `LogicalVolume` può essere posizionato in punti diversi: tutta la gerarchia di sotto volumi viene anch'essa posizionata in ogni copia. Ogni `PhysicalVolume` è identificato da un parametro chiamato *copy number* impostabile dall'utente e utilizzato per referenziarlo in modo univoco.

La costruzione delle tre parti del rivelatore avviene in tre metodi.

ConstructTelescope In questo metodo vengono costruiti i tre piani del telescopio a silicio. In primo luogo viene costruito il solido che rappresenta un'intero piano e viene posizionato 3 volte con 90 mm di separazione lungo l'asse z. E' sufficiente creare un `G4VSolid` e un `LogicalVolume` e posizionarlo tre volte.

```
//***************  DetectorConstrucion.cc   ************
G4VPhysicalVolume* DetectorConstruction::ConstructTelescope()
{

G4double halfSensorSizeX = noOfSensorStrips*teleStripPitch
    /2.;
G4double halfSensorSizeY = sensorStripLength/2.;
G4double halfSensorSizeZ = sensorThickness/2.;

// Si definisce la geometria del piano del telescopio
G4Box * solidSensor = new G4Box("Sensor",
  halfSensorSizeX,halfSensorSizeY,halfSensorSizeZ);

// e il suo Logical volume
G4LogicalVolume * logicSensorPlane = new G4LogicalVolume(
    solidSensor, // its solid
    silicon,  //its material
    "SensorPlane"); //its name

// ora si posiziona il logical volume tre volte
physiFirstSensor = new G4PVPlacement(0, //no rotation
    posFirstSensor,
    logicSensorPlane,    //its logical volume
    "FirstSensor",       //its name
    logicWorld,        //its mother  volume
    false,           //no boolean operation
    0);            //copy number
// 2nd and 3rd Plane of Si tracker
[...]
```

E' necessario definire una singola strip con un solido e un `LogicalVolume` . Questa viene poi posizionata all'interno dei piani del telescopio utilizzando una `G4VReplica`: questo oggetto è un particolare tipo di `G4VPlacement` che permette di ripetere un `LogicalVolume` all'interno di un altro lungo un asse. E' sufficiente posizionare la strip 600 volte all'interno del `LogicalVolume` del piano e tutti i piani del telescopio acquiseranno la stessa struttura.

```
//*********  DetectorConstruction::ConstructTelescope()
    ********
G4Box * solidSensorStrip =
new G4Box("SensorStrip",  halfSensorStripSizeX,
    halfSensorStripSizeY,halfSensorStripSizeZ);
```

```
4
5 G4LogicalVolume * logicSensorStrip =
6 new G4LogicalVolume(solidSensorStrip,silicon,"SensorStrip");
7
8 physiSensorStrip = new G4PVReplica("SensorStrip",    //its
    name
9     logicSensorStrip,    //its logical volume
10    logicSensorPlane,    //its mother
11    kXAxis,                //axis of replication
12    noOfSensorStrips,    //number of replica
13    teleStripPitch);       //witdth of replica
```

ConstructEMCalo Il metodo `ConstructEMCalo()` utilizza un oggetto `G4Box` per creare due cristalli, entrambi di PbWO4, uno all'interno dell'altro. Entrambi hanno la stessa lunghezza e lo stesso centro, ma hanno dimensioni diverse. Il cristallo centrale ha dimensioni 22m x 22m x 230mm, mentre quello esterno ha dimensioni 110mm x 110mm x 230mm. Come nel caso delle strip di silicio, il cristallo centrale viene posizionato all'interno di quello esterno utilizzando la relazione gerarchica tra `LogicalVolumes`.

ConstructHadCalo Il metodo `ConstructHadCalo()` costruisce il calorimetro adronico in due fasi. Prima di tutto viene creato un `G4Tubs` di ferro contenente l'intero calorimetro e viene posizionato nel `WorldVolume`. In seguito vengono inseriti gli strati sensibili di argon liquido creando un singolo volume, uno strato, e replicandolo all'interno del volume creato in precedenza. Non viene utilizzata una replica ma viene calcolata la posizione di ogni strano e viene creato un `G4PVPlacement` all'interno del volume principale del calorimentro. Da notare che il copy number di ogni strato è diverso, fatto necessario per identificare il punto di interazione delle particelle nel detector.

```
1 //*********  DetectorConstruction::ConstructHadCalo()
    *********
2 //We now make layers of LAr and add them to the hadronic calo
    logic
3 G4Tubs* hadLayerSolid = new G4Tubs( "HadCaloLayerSolid",
4     0   , //innerradius
5     hadCaloRadius , //outeradium
6     hadCaloLArThickness/2, //half lenght in Z
7     0,
8     CLHEP::twopi);
9 G4LogicalVolume* hadLayerLogic = new G4LogicalVolume(
10    hadLayerSolid,
11    lar,
12    "HadLayerLogic");//its name
13
```

```
G4ThreeVector absorberLayer(0,0,hadCaloFeThickness);
G4ThreeVector activeLayer(0,0,hadCaloLArThickness);
G4int layerCopyNum = hadCaloCopyNum;
for (int layerIdx = 0 ; layerIdx < hadCaloNumLayers ; ++
    layerIdx)
{
  G4ThreeVector position = (layerIdx+1)*absorberLayer + (
   layerIdx+0.5)*activeLayer;
  // La posizione deve essere relativa al volume logico madre
  position -= G4ThreeVector(0,0,halfHadCaloHalfZ);
  new G4PVPlacement(0,
      position,
      hadLayerLogic,
      "HadCaloLayer",
      hadCaloLogic,  //the mother volume is the Fe tubs
      false,
      ++layerCopyNum  //1001+layerIndex
  );
}
```

1.2.3 Physics List

La classe `PhysicsList` che deriva 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. L'utente deve ridefinire obbligatoriamente costruttore, distruttore e i metodi `ConstructParticel()`, `ConstructProcess()` e `SetCuts()`. All'interno del framework sono disponibili diverse PhysicsLists preimpostate che l'utente può combinare all'interno della sua classe: in questo esercizio viene usata un'istanza di `G4EmStandardPhysics`, la `PhysicsList` standard per i processi elettromagnetici. All'interno di `Construct Particle` è sufficiente quindi chiamare la funzione

```
emPhysicsList->ConstructParticle();
```

per attivare tutte le particelle previste da questa `PhysicsList` che comprende elettroni, muoni e rispettive antiparticelle e alcuni hadroni (pioni, protoni, kaoni, particelle alfa). Se l'utente lo desidera può attivare direttamente le singole particelle con le seguenti istruzioni.

```
#include "G4ParticleTypes.hh"
void PhysicsList::ConstructParticle()
{
  G4Electron::Electron();
  G4Positron::Positron();
}
```

La `PhysicsList` contiene inoltre la definizioni dei processi attivi. Un processo è un tipo di interazione tra le particelle definite dalla `PhysicsList` e i materiali, ad esempio l'effetto Compton, fotoelettrico etc. I processi vanno instanziati all'interno del metodo `ConstructProcess()`. E' sempre necessario definire il processo di trasporto delle particelle, inoltre si deve chiamare la funzione `ConstructProcess()` di ogni `PhysicsList` preimpostata che si vuole utilizzare.

```
void PhysicsList::ConstructProcess()
{
  AddTransportation();
  emPhysicsList->ConstructProcess();
}
```

La funzione `SetCuts()` infine imposta i limite di range (quindi di energia) sotto i quali non vengono create tracce secondarie. In questo esercizio si imposta nel costruttore della `PhysicsList` il valore di 10 μm e si chiama la funzione `SetCutsWithDefault()`, ma è possibile impostare diversi cut per diversi tipi di particelle e diverse regioni del detector.

1.2.4 PrimaryGeneratorAction

La classe `PrimaryGeneratorAction` deriva da `G4VUserPrimaryGeneratorAction` e permette all'utente di impostare le caratteristiche delle particelle iniziali in un evento: tipo di particella, energia, direzione del moto. E' necessario implementare il costruttore e la funzione `GeneratePrimaries(G4Event*)`: Geant4 mette a disposizione due utili classi `G4ParticleGun` e `G4GeneralParticleSource` per generare gli eventi e impostare facilmente le varie caratteristiche del fascio. E' possibile definire una distribuzione angolare e di energia delle particelle iniziali e anche generare più particelle alla volta.

In questo esercizio si è definita la funzione `PrimaryGeneratorAction::InitializeGPS()` per impostare un `GeneralParticleSource` .

```
// ********** PrimaryGeneratorAction.cc
   **********************
G4VPrimaryGenerator* PrimaryGeneratorAction::InitializeGPS()
{
  G4GeneralParticleSource * gps = new G4GeneralParticleSource
   ();

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

```
10
11   // set energy distribution
12   G4SPSEneDistribution *eneDist = gps->GetCurrentSource()->
      GetEneDist() ;
13   eneDist->SetEnergyDisType("Mono"); // or gauss
14   eneDist->SetMonoEnergy(2.0*GeV);
15
16   // set position distribution
17   G4SPSPosDistribution *posDist = gps->GetCurrentSource()->
      GetPosDist();
18   posDist->SetPosDisType("Beam");  // or Point,Plane,Volume,
      Beam
19   posDist->SetCentreCoords(G4ThreeVector(0.0*cm,0.0*cm,-80.0*
      cm));
20   posDist->SetBeamSigmaInX(0.1*mm);
21   posDist->SetBeamSigmaInY(0.1*mm);
22
23   // set angular distribution
24   G4SPSAngDistribution *angDist = gps->GetCurrentSource()->
      GetAngDist();
25   angDist->SetParticleMomentumDirection( G4ThreeVector(0.,
      0., 1.) );
26   angDist->SetAngDistType("beam2d");
27   angDist->SetBeamSigmaInAngX(0.1*mrad);
28   angDist->SetBeamSigmaInAngY(0.1*mrad);
29   angDist->DefineAngRefAxes("angref1",G4ThreeVector
      (-1.,0.,0.));
30
31   return gps;
32 }
```

In questo esercizio si utilizzano pioni positivi monoenergetici da 2 GeV distribuiti in un'area circolare con $\sigma = 0.1\,mm$ sia su x che su y, con una distribuzione angolare bidimensionale con $\sigma = 0.1\,mrad$.

La generazione degli eventi è affidata al metodo `GeneratePrimaries(G4Event*)`.

```
1 void PrimaryGeneratorAction::GeneratePrimaries(G4Event*
    anEvent)
2 {
3   // gun is a private variabile corrisponding to the
4   // generator loaded by InitializeGPS()
5   gun->GeneratePrimaryVertex(anEvent);
6 }
```

Le caratteristiche del fascio possono essere modificate facilmente attraverso i comandi delle macro di Geant4, sotto la voce `/gps/[particle|energy|pos|ang]`

1.3 Macro

Come visto nel modulo `Main` 1.2.1 a pagina 1 una simulazione Geant4 viene eseguita dall'utente passando al componente `G4UIManager` una macro contenente una serie di comandi. Nella macro, oltre a gestire la visualizzazione, è possile ad esempio cambiare le impostazioni del `GenericParticleSource` per la generazione delle particelle primarie. Come vedremo in seguito molti componenti di Geant4 possono essere infatti sia configurati all'interno del codice o direttamente con comandi macro: è sufficiente creare delle apposite classi per gestire ad esempio i dettagli della geometria del detector e quindi poter modificare diversi parametri durante l'esecuzione di una macro.

```
# Sets some default verbose
/control/verbose 2
/run/verbose 2

# Create a scene handler for a specific graphics system
/vis/open OGLSQt 600x600-0+0

# draw scene
/vis/drawVolume

/vis/viewer/set/viewpointThetaPhi 70 180 deg
#/vis/viewer/zoom 1.2

# for drawing the tracks
/vis/scene/add/trajectories

# for drawing the hits
/vis/scene/add/hits

# (if you prefer refreshing each event, comment out next line
    )
#/vis/scene/endOfEventAction accumulate

/vis/scene/add/axes 10 0 0 10 cm

# It's possible to run the simulation directly from the macro
/run/beamOn 10
```

1.4 Risultato

In questo esercizio è stata simulata l'interazione del detector con un fascio di π^+ e di e^- da 2 GeV. Notare come i pioni interagiscano principalmente con il calorimetro adronico, mentre gli elettroni con quello elettromagnetico.

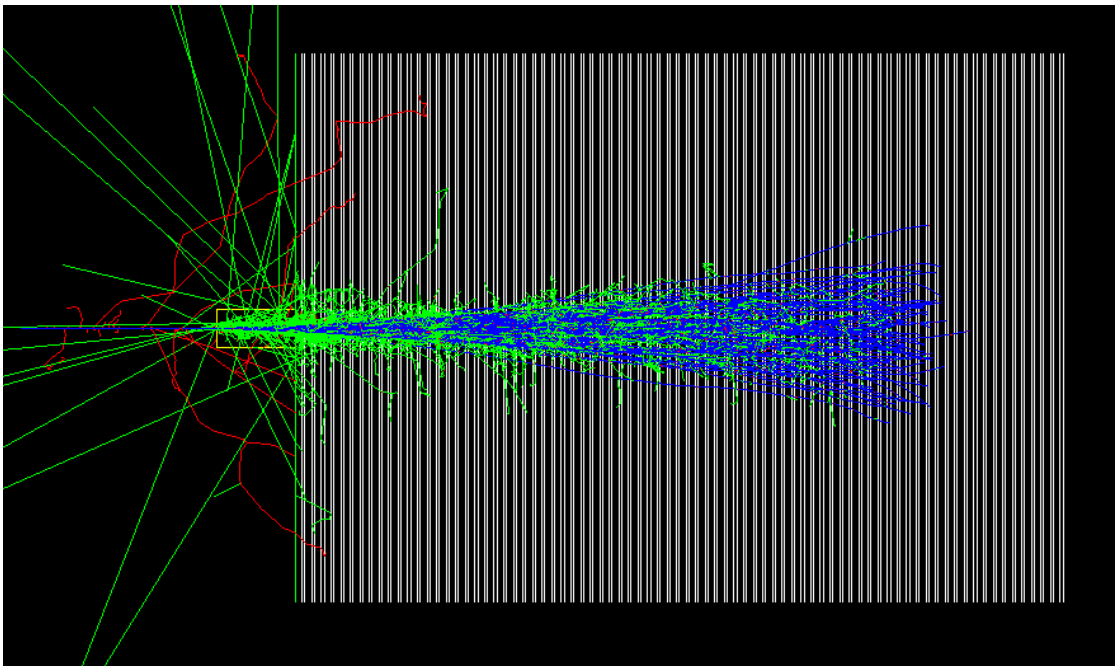

Figura 1.1: π^+ da 2 GeV che interagiscono principalmente con il calorimentro adronico

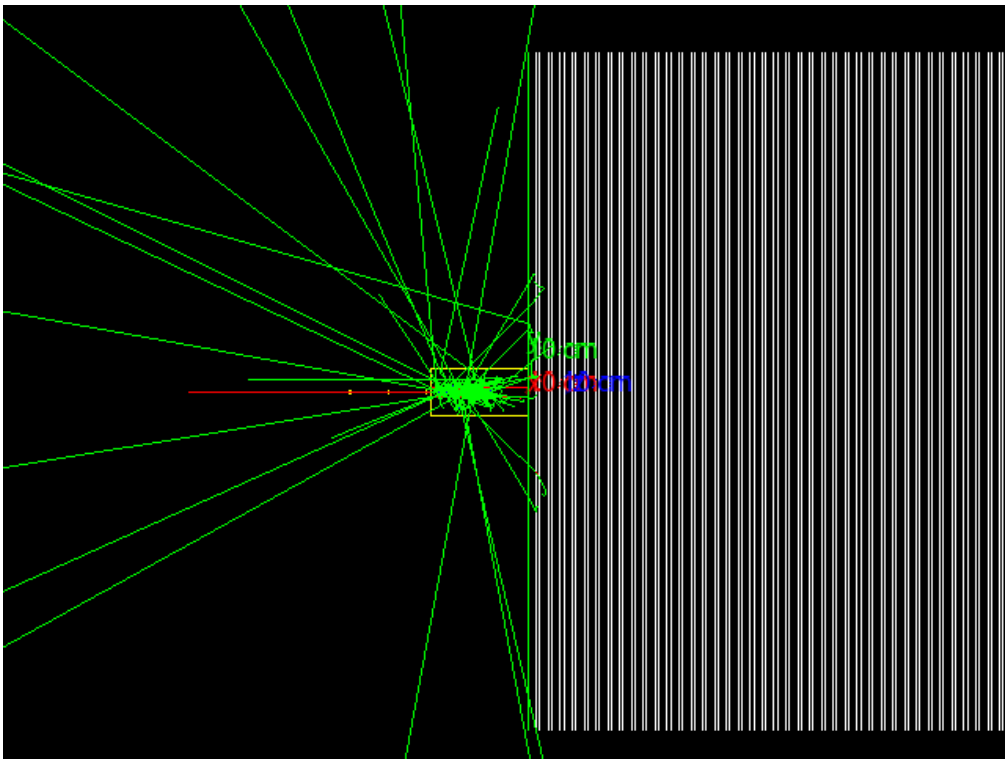

Figura 1.2: Singolo e^- da 2 GeV che viene completamente assorbito dal calorimetro elettromagnetico.

Capitolo 2

Esercizio 2

2.1 Descrizione generale

Questo esercizio consiste nella simulazione di un telescopio a tre livelli, già incontrato nel primo esercizio. Il primo e l'ultimo livello sono piani formati da strip di silicio, il livello intermedio può essere uguale agli altri o un detector under test (DUT) caratterizzato da un pitch delle strip di 50 μm invece che di 20 μm. Il secondo livello può essere ruotato sull'asse Y.

Inoltre in questo esercizio vengono mostrati gli strumenti per analizzare l'interazione (`Hit`) delle particelle con le parti sensibili del detector (`SensitiveDetector`) e viene simulata anche l'elettronica di lettura per produrre i dati finali detti `Digi`.

Infine viene definita un'interfaccia utente per il detector attraverso la classe `DetectorMessenger` che permette di controllare dalla macro la posizione del secondo livello del telescopio e la presenza del DUT.

2.2 Componenti del programma

2.2.1 DetectorConstruction

La geometria del telescopio è analoga a quella costruita nella sezione 1.2.2 a pagina 6: si utilizza una `G4PVReplica` per replicare il volume di una singola strip nel volume principale di un piano del telescopio. Il secondo livello è sostituito da un DUT se la variabile `isSecondPlaneDUT` è `true`, ma l'unico cambiamento è il pitch delle singole strip.

In questo esercizio è possibile modificare alcuni parametri del detector attraverso getter/setter del `DetectorConstruction` che verrano chiamati dal `DetectorMessenger` per modificare i valori attraverso una macro. Ovviamente in caso di modifiche è necessario aggiornare la geometria del detector quindi il `DetectorConstruction` deve implementare una funzione che si occupa di pulire i dati relativi alla geometria e di ricostruirla.

```
1 // ************** DetectorConstruction.cc ***********
```

```
void DetectorConstruction::UpdateGeometry()
{
  // Cleanup old geometry
  G4GeometryManager::GetInstance()->OpenGeometry();
  G4PhysicalVolumeStore::GetInstance()->Clean();
  G4LogicalVolumeStore::GetInstance()->Clean();
  G4SolidStore::GetInstance()->Clean();

  G4RunManager::GetRunManager()->DefineWorldVolume(Construct
   ());
}
```

2.2.2 DetectorMessenger

La classe `DetectorMessenger` deriva da `G4UImessenger` e definisce alcuni comandi utilizzabili dall'interfaccia interattiva di Geant4 o nelle macro per modificare i parametri del detector.

Ad esempio se si vuole ruotare di un angolo θ il secondo piano del telescopio o attivare il DUT si possono implementare questi comandi.

```
// ************** DetectorMessenger.cc ****************
DetectorMessenger::DetectorMessenger(DetectorConstruction *
   det)
:detector(det)
{
detDir = new G4UIdirectory("/det/");
detDir->SetGuidance("detector construction commands");

secondSensorDir = new G4UIdirectory("/det/secondSensor/");
secondSensorDir->SetGuidance("comands related to the second
   sensor plane");

thetaCmd = new G4UIcmdWithADoubleAndUnit ("/det/secondSensor/
   theta", this);
thetaCmd->SetGuidance("Select rotation angle of second sensor
    plane around y axis");
thetaCmd->SetParameterName("thetaDUT",true);
thetaCmd->SetUnitCategory("Angle");
thetaCmd->SetDefaultUnit("deg");

setDUTsetupCmd = new G4UIcmdWithABool("/det/secondSensor/
   DUTsetup",this);
setDUTsetupCmd->SetGuidance("Select setup. true to have DUT (
   Device Under Test) setup: second Si plane replaced by DUT"
   );
setDUTsetupCmd->AvailableForStates(G4State_Idle);
```

```
20
21 // Comando importante! Aggiorna la geometria
22 updateCmd = new G4UIcmdWithoutParameter("/det/update",this);
23 updateCmd->SetGuidance("force to recompute geometry.");
24 updateCmd->SetGuidance("This command MUST be applied before
       \"beamOn\" ");
25 updateCmd->SetGuidance("if you changed geometrical value(s)."
       );
26 updateCmd->AvailableForStates(G4State_Idle);
27 }
```

E' necessario ovviamente definire anche una funzione che esegua i comandi chiamando i setter del `DetectorConstruction` .

```
1 // ************** DetectorMessenger.cc ****************
2 void DetectorMessenger::SetNewValue(G4UIcommand* command,
       G4String newValue)
3 {
4   if ( command == thetaCmd )
5   detector->SetDUTangle(thetaCmd->GetNewDoubleValue(newValue)
       );
6
7   if ( command == setDUTsetupCmd )
8   detector->SetDUTSetup( setDUTsetupCmd->GetNewBoolValue(
       newValue) );
9
10  if ( command == updateCmd )
11  detector->UpdateGeometry();
12 }
```

2.2.3 SensitiveDetector

Alcuni parti del detector sono considerate *attive* cioè in grado di registrare informazioni sulla particella che li attraversa. Ad esempio un cristallo scintillatore o l'argon liquido in un calorimetro a sampling sono materiali attivi: nella simulazione è necessario registrare ad esempio il punto di interazione o la quantità di energia depositata dalla particella. Queste informazioni raggruppate vengono chiamate `Hit` e caratterizzano una singola interazione della particella con il materiale. Il componente di Geant4 che svolge questo compito si chiama `SensitiveDetector`. L'utente deve implementare una classe che rappresenti le informazioni contenute in un `Hit` e creare un oggetto che derivi da `G4VSensitiveDetector`: il `SensitiveDetector` deve essere registrato nei `LogicalVolume` da monitorare e la funzione `SensitiveDetector::ProcessHits` verrà chiamata ogni volta che una particella interagisce con il materiale.

Nel `DetectorConstruction` terminata la costruzione dei volumi viene instanziato e registrato il `SensitiveDetector` .

```
// ******* DetectorConstruction::ConstructTelescope() ****
static SensitiveDetector* sensitive = 0;
if ( !sensitive) {
  sensitive = new SensitiveDetector("/myDet/SiStripSD");
  //We register now the SD with the manager
  G4SDManager::GetSDMpointer()->AddNewDetector(sensitive);
}
// Il SensitiveDetector va associato al LogicVolume delle
   strip
logicSensorStrip->SetSensitiveDetector(sensitive);
```

La funzione `SensitiveDetector::ProcessHits` viene chiamata ad ogni interazione ed estrae le informazioni desiderate dal `G4Step`, un oggetto che rappresenta un singolo step di simulazione della particella all'interno del materiale. In ogni `Hit` si memorizza l'id della strip, il piano del telescopio e l'energia depositata, inoltre è possibile capire se la particella è primaria o secondaria. Ogni `Hit` viene poi salvato nell'`HitsCollection` dell'evento in corso per poter essere riutilizzato successivamente. I dettagli delle informazioni estraibili da un `G4Step` saranno presentati nell'esercizio 3a (vedi 3.2.3 a pagina 25).

2.2.4 RunAction e EventAction

Geant4 mette a disposizione dell'utente degli utili callbacks che vengono chiamati prima e dopo i diversi passi della simulazione. L'utente può implementare le classi `RunAction` e `EventAction` rispettivamente per eseguire del codice prima e dopo ogni run (insieme di eventi sotto le stesse condizioni del detector) o prima e dopo ogni evento. Questi oggetti vanno registrati nel `RunManager` prima di inizializzare il kernel.

```
// ************* Main.cc ****************
EventAction* event_action = new EventAction;
runManager->SetUserAction(event_action);

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

In questo esercizio i callback `BeginOfRunAction` e `EndOfRunAction` vengono utilizzati per creare e chiudere il `TTree` per il salvataggio dei dati di ogni run. La classe `EventAction` gestisce invece l'elaborazione degli `Hit` alla fine di ogni evento attraverso la classe `SiDigitizer` spiegata di seguito.

2.2.5 SiDigitizer

Gli `Hit` in un detector non corrispondono quasi mai alle informazioni di read-out reale di un detector: il processo di misura passa sempre attraverso degli strumenti

elettronici analogici e/o digitali in grado di elaborare e raccogliere le informazioni estratte dai materiali attivi del detector. Ad esempio la luce di scintillazione di un cristallo di tungstato di piombo produce un impulso di corrente al fotocatodo di un fotomoltiplicatore che viene poi analizzato da una catena di lettura fino al salvataggio. Gli `Hit` rappresentano le quantità fisiche della risposta del detector (energia accumulata), i `Digi` rappresentano l'informazione elaborata (ADC counts).

In Geant4 è possibile definire dei `Digitizer` (che derivano da `G4VDigitizer Module`) per elaborare le `HitsCollection` di ogni evento e produrre delle `Digi Collection` . In questo modo è possibile simulare il rumore elettronico, i piedistalli di energia del detector, la conversione energia/carica. In questo esercizio è stato simulato l'effetto di crosstalk di un'insieme di strip di silicio, cioè la perdita di una piccola frazione di carica di una strip a favore delle adiacenti.

2.2.6 RootSaver

`RootSaver` è una semplice classe helper che si occupa di salvare su `TTree` le informazioni degli `Hit` e dei `Digi` prodotte in ogni evento. Vengono create diverse branch nel TTree e viene inserita una entry per ogni evento. In particolare si salvano i valori della carica presente in ogni strip (dai `Digi`), l'energia accumulata in ogni piano del telescopio (`Hit`), la strip in cui è passata la traccia principale in ogni piano (`Hit`) e i parametri del fascio (posizione, angolo).

L'oggetto viene inizializzato dalla `BeginOfRunAction` ,i dati vengono passati per il salvataggio alla fine di ogni evento dalla `EndOfEventAction` e infine il TTree viene salvato su file alla fine del run dalla `EndOfRunAction` .

2.3 Risultato

2.3.1 Telescopio semplice (senza DUT)

Nella prima parte dell'esercizio si utilizza il più semplice `G4ParticleGun` invece del `GenericParticleSource` . Si vuole produrre un fascio di π^+ da 2 GeV in due diverse configurazioni: collimato perfettamente nell'origine degli assi o distributo in modo randon in un rettangolo $0.1 \cdot 2$ mm.

La `PhysicsList` e la macro del esercizio 2a sono identiche a quelle del esercizio 1b. Si analizzano i dati salvati dal `RootSaver` : la carica raccolta da ogni strip dei tre piani e l'energia depositata in ogni piano.

Il DUT è disattivato e sarà oggetto della Esercizio 2b.

Fascio collimato

Per produrre un fascio collimato è necessario generare eventi nel `PrimaryGenerationAction` nel modo seguente.

```
1 G4double x0 = 0.*cm, y0 = 0.*cm, z0= 0.0*cm;
```

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

I tre plot mostrano la carica per ogni strip nei tre piani del telescopio estratta dai `Digi` . Viene simulato un rumore gaussiano ($\sigma = 1000$) su un piedistallo di 5000 conteggi. La posizione di passaggio della particella è estraibile dal picco, che corrisponde a un'energia depositata di $Q \cdot 3.6$ eV (il fattore di conversione utilizzato nel `SiDigitizer`). Eseguendo la simulazione a 2 GeV e a 20 Gev si può notare come i pioni meno energetici interagiscano di più con il detector (depositano più energia) e vengano quindi deviati maggiormente.

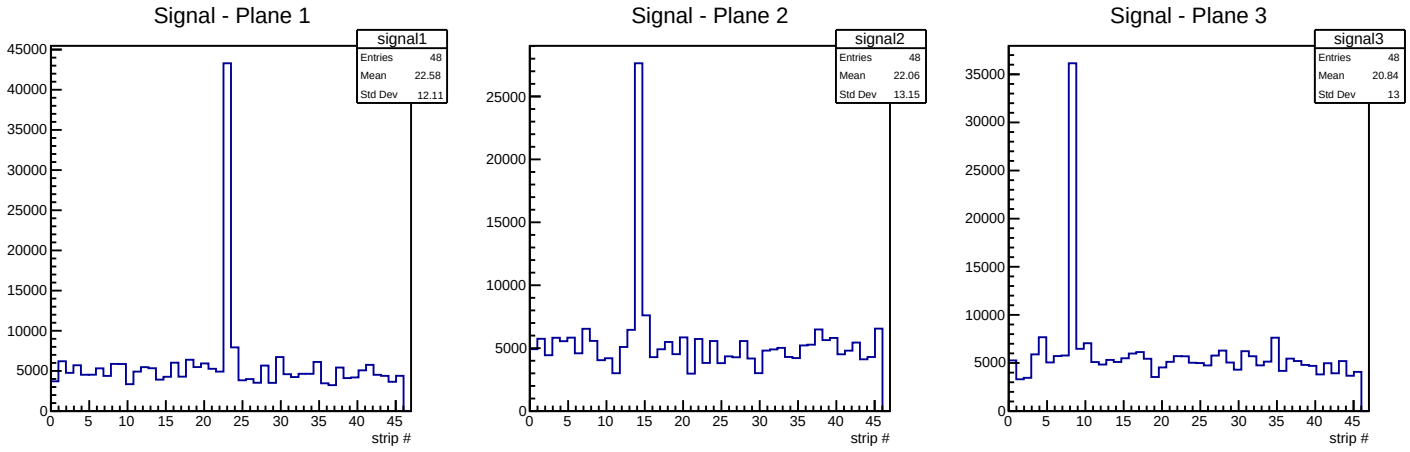

Figura 2.1: Carica per strip sui tre piani del telescopio con fascio di π^+ da 2 GeV

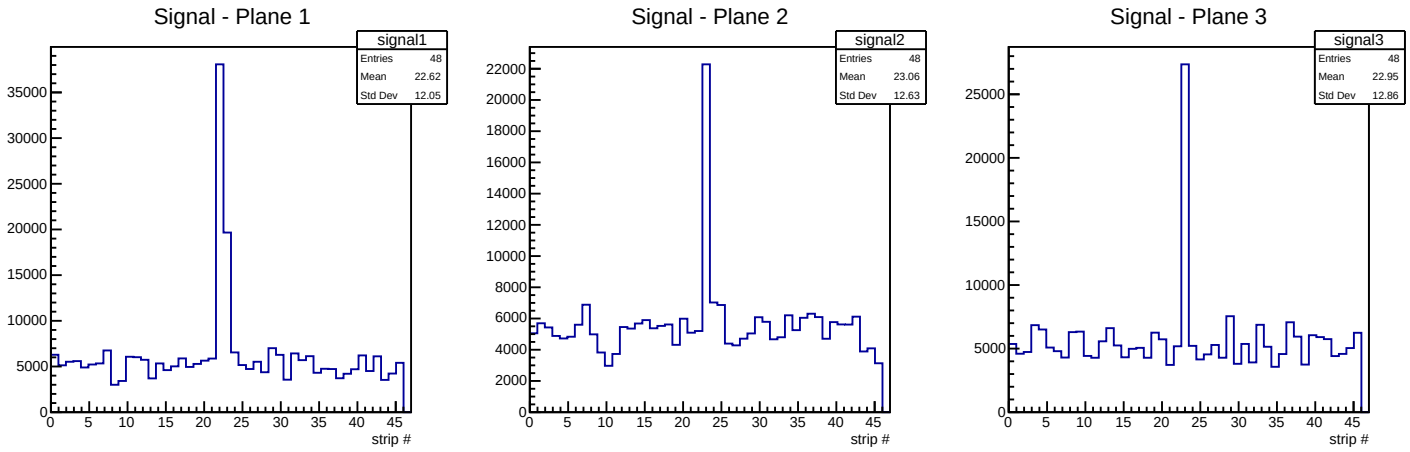

Figura 2.2: Carica per strip sui tre piani del telescopio con fascio di π^+ da 20 GeV

La figura 2.3 nella pagina successiva mostra l'energia depositata sul primo piano del telescopio da un fascio collimato di 10000 π^+ da 2 GeV. Si nota che la distribuzione non è gaussiana, ma asimmetrica con un coda a energia maggiore:

questo è il comportamento atteso per l'energia depositata dalle particelle cariche in uno strato sottile ed è modellizzato dalla distribuzione di Landau.

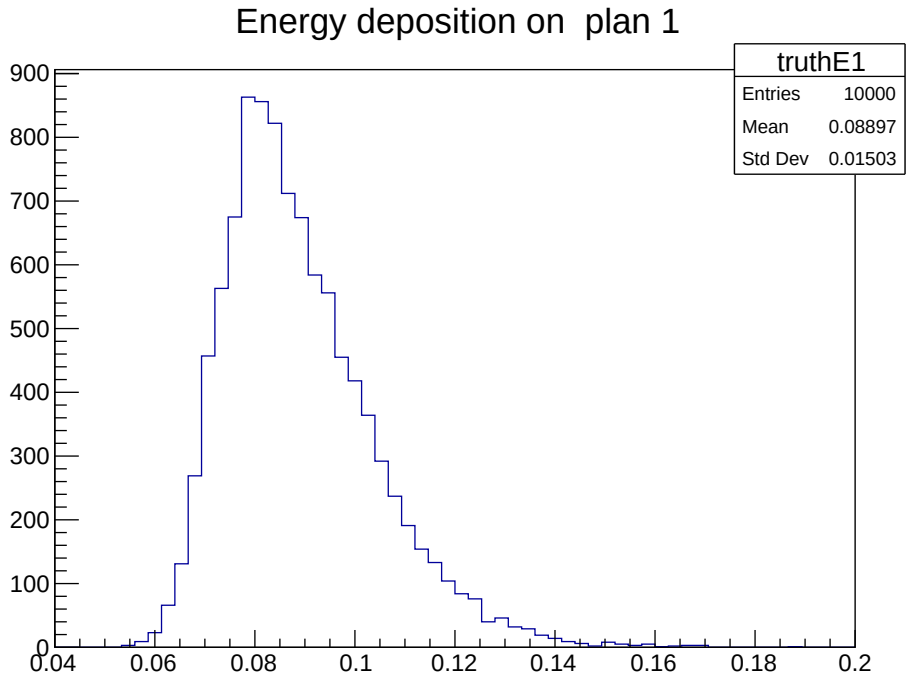

Figura 2.3: Energia depositata sul primo piano del telescopio

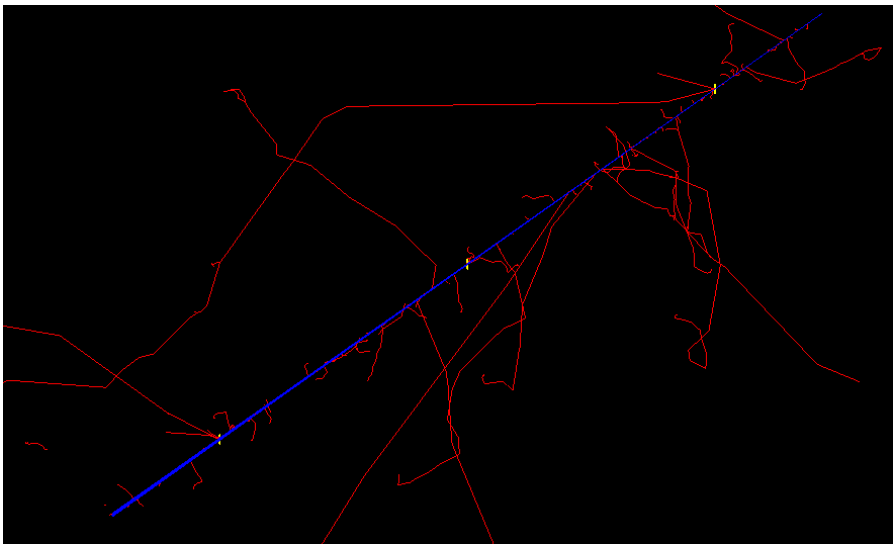

Figura 2.4: I tre piani del telescopio sono in giallo, in rosso le tracce degli elettroni (raggi delta)

Fascio rettangolare

Per produrre un fascio con distribuzione uniforme in un rettangolo di 0.1 per 2 mm sul piano xy si imposta il `G4ParticleGun` come segue.

```
G4double z0 = 0.*mm, x0 = 0.*mm, y0 = 0.*mm;
x0 = -0.05 + 2*0.05*G4UniformRand();
y0 = -1.0+ 2*G4UniformRand();
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);
```

In figura 2.5 si mostra la distribuzione di carica prodotta da 1000 π^+ da 2 GeV sul primo piano del telescopio corrispondente a particelle primarie generate secondo la distribuzione uniforme impostata.

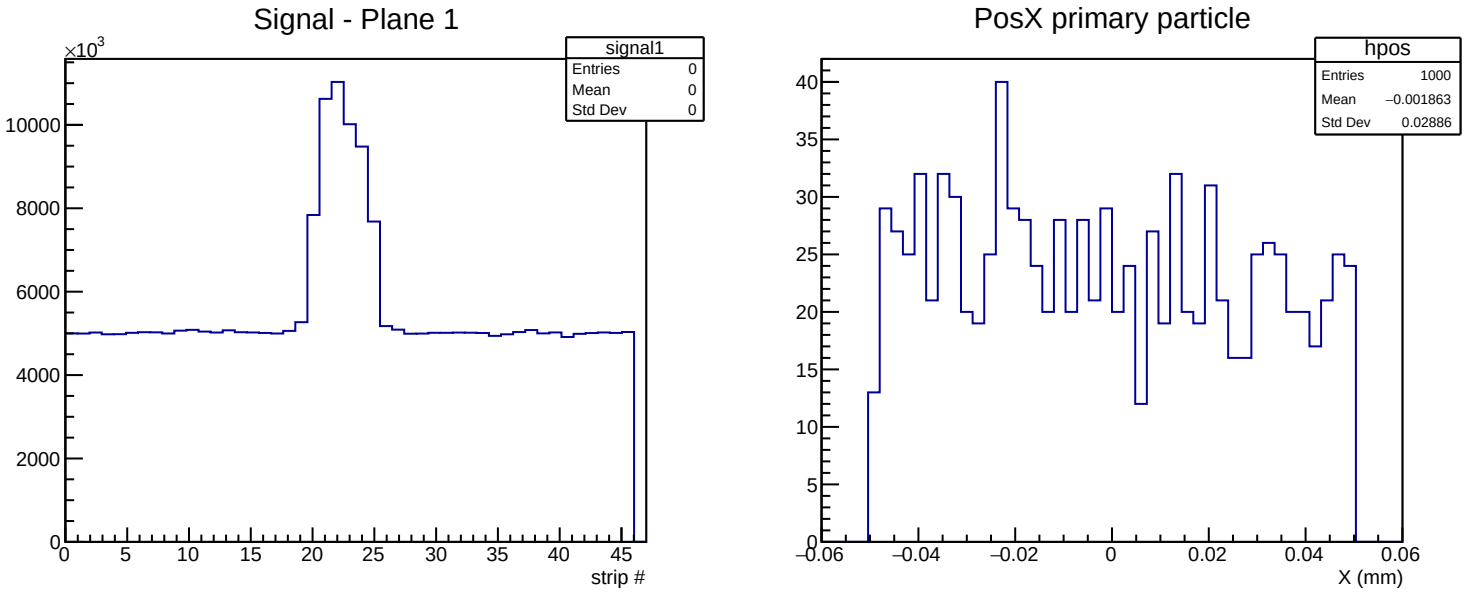

Figura 2.5: Signale sul primo piano e distribuzione della posizione X della particella primaria

2.3.2 Telescopio con DUT

Nella seconda parte dell'esercizio attiviamo il DUT attraverso i comandi macro definiti nella sezione 2.2.2 a pagina 14:

```
# Activate the DUT setup
```

```
/det/secondSensor/DUTsetup
# Update the detector geometry
/det/update
```

Inoltre modifichiamo il `PrimaryGeneratorAction` per utilizzare un `GenericParticleSource` invece di un `G4ParticleGun` in modo tale da poter controllare in maniera sofisticata il fascio dall'interfaccia di Geant4 attraverso i comandi `/gps` . Si utilizza un fascio di 1000 π^+ e μ^- da 2 GeV.

```
# Seleziona il tipo di particella
/gps/particle pi+
# oppure
/gps/particle mu-

# Impost energia, posizione e direzione
/gps/energy 2. GeV
/gps/position 0. 0. 0. m
/gps/direction 0. 0. 1.

 # Beam di forma gaussiana
/gps/pos/type Beam
/gps/pos/sigma_x 0.1 mm
/gps/pos/sigma_y 0.1 mm

# Distribuzione angolare gaussiana
/gps/ang/type beam2d
/gps/ang/sigma_x 0.1 mrad
/gps/ang/sigma_y 0.1 mrad
/gps/ang/rot1 -1. 0. 0.

# Lancia simulazione di 1000 eventi
/run/beamOn 1000
```

Si effettua una scan sull'angolo theta del DUT utilizzando `/control/foreach` per eseguire una macro con diversi parametri.

```
/control/foreach dutsetupRotateOnce.mac angle  0 10 20 45
```

```
# ****************** dutsetupRotateOnce.mac ***************

# Impostazione dei parametri del fascio (come sopra)
[...]

# Si imposta l'angolo utilizzando il parametro {angle}
/det/secondSensor/DUTsetup true
```

```
/det/secondSensor/theta {angle} deg
# Si imposta anche il parametro di crosstalk
/det/digi/crosstalk 0.05
# Aggiorna il detector
/det/update

/run/beamOn 1000
```

In figura 2.6 viene mostrata l'energia depositata sul DUT a seconda dell'angolo: si nota che all'aumentare dell'angolo la media della distribuzione aumenta poichè le particelle possono interagire con uno spessore maggiore del detector. Questo tipo di simulazione è utile per studiare gli effetti del non perfetto allineamento di un detector sui dati raccolti.

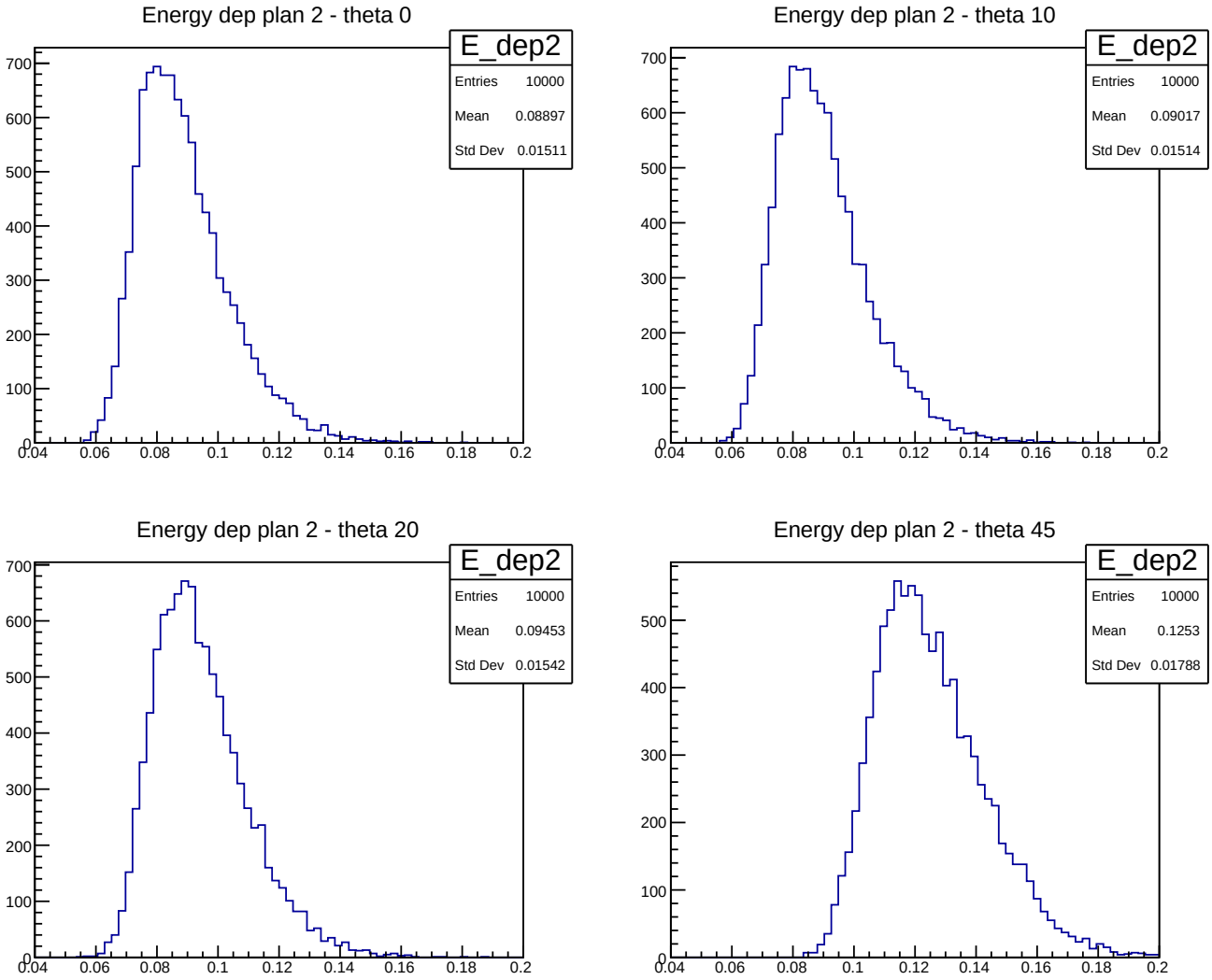

Figura 2.6: Energia depositata sul DUT per diversi angoli rispetti all'asse y da fascio di π^+ di 2 GeV

Capitolo 3

Esercizio 3a

3.1 Descrizione generale

Questo esercizio consiste nella simulazione di un telescopio a tre piani a strip di silicio e di un calorimetro elettromagnetico. I due detector sono costruiti come nell'esercizio 1, ma le strip di silicio sono 600 per ogni piano.

Lo scopo di questo esercizio è di utilizzare ulteriori `UserActions` rispetto a quelle viste nell'esercizio 2 (cfr 2.2.4 a pagina 16) per registrare informazioni durante la simulazione: in particolare invece del meccanismo degli `Hit` e `SensitiveDetector` si utilizzeranno le `SteppingAction` e `StackingAction` per analizzare gli step della simulazione e la creazione delle tracce.

3.2 Componenti del programma

3.2.1 Analysis

Il salvataggio dei dati è affidato a una classe helper: `Analysis` . Si utilizza il metodo del singleton per rendere disponibile una singola istanza dell'oggetto a tutta l'applicazione.

```
1  // ************ Analysis.cc **********
2  Analysis* Analysis::singleton = 0;
3
4  Analysis* Analysis::GetInstance() {
5    if ( singleton == 0 ) {
6      static Analysis analysis;
7      singleton = &analysis;
8    }
9    return singleton;
10 }
```

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

PrepareNewRun Questa funzione viene chiamata da `RunAction::BeginOfRun Action` e viene utilizzata per inizializzare l'analisi dei dati: vengono create diverse variabili per le statistiche complessive a livello di run e vengono instanziati gli istogrammi necessari.

PrepareNewEvent Questa funzione viene chiamata da `EventAction::BeginOf EventAction` e viene utilizzata per azzerare le statistiche riguardanti un singolo evento quali energia totale depositata, numero di tracce secondarie.

EndOfRun Alla fine di ogni run `RunAction::EndOfRunAction` chiama questa funzione che si occupa di stampare alcuni dati riassuntivi (numero medio di gamma, e^+, e^-, energia depositata) e di salvare gli istogrammi generati su `TFile` di `ROOT` .

EndOfEvent Quanto viene chiamata da `EventAction::EndOfEventAction` questa funzione riempie gli istogrammi con i dati relativi all'evento.

AddSecondary Questa funzione viene chiamata dalla classe `StackingAction` (che verrà descritta in seguito) per memorizzare la creazione di una nuova traccia secondaria (numero e tipo di particella)

AddEDepEM Questa funzione viene chiamata dalla `SteppingAction::User SteppingAction` passando energia depositata, coordinata z e `copynumber` del detector per ogni step della simulazione. Queste informazioni vengono salvate e viene riempito l'istogramma dell'energia depositata lungo la coordinata z del calorimetro.

SetBeam La `StackingAction` raccolglie anche informazioni sulla traccia primaria dell'evento che vengono salvate attraverso questa funzione. Si salva il tipo di particelle e l'energia.

3.2.2 StackingAction

Una `Track` rappresenta un'instantanea della particella durante la simulazione: contiene le informazioni sulla sua posizione, la sua energia, il momento, il tipo di particella e i processi a cui può essere sottoposta. Lo stato di una particella, la `G4Track` , viene modificato attraverso i `G4Step` caratterizzati da un punto di inizio e fine, durata, differenza di energia e momento. Una `Track` non è una collezione di `Step` , ma viene aggiornata da essi. Durante la simulazione una traccia può essere terminata a a causa di un decadimento o di un completo assorbimento e tracce secondarie possono essere create (ad esempio delta-rays sopra la soglia di range definita da Geant4).

Oltre alla possibilità di monitorare l'inizio e la fine di un run o di un evento, Geant4 permette all'utente di gestire la priorità nella simulazione delle tracce primarie e secondarie attraverso la `UserStackingAction` . Geant4 utilizza lo `StackingManager` per suddividere le tracce in diversi stack o code last-in-first-out: lo stack *Urgent* contiene le tracce in corso di simulazione, quello *Waiting* contiene

le tracce che diventeranno *Urgent* quando quello stack sarà vuoto. Quando una nuova Track viene creata l'utente può decidere in che coda inserirla attraverso il metodo `ClassifyNewTrack` della classe `UserStackingAction` .

In questo esercizio non si gestisce in modo particolare la priorità di simulazione, ma si utilizza la funzione `ClassifyNewTrack` per salvare informazione sulle tracce create.

```
// *********** StackingAction.cc ***************
G4ClassificationOfNewTrack
StackingAction::ClassifyNewTrack( const G4Track * aTrack )
{
  // always "urgent" in current applications
  G4ClassificationOfNewTrack result( fUrgent );

  if ( aTrack->GetParentID() > 0 ){ //This is a secondary
    Analysis::GetInstance()->AddSecondary(aTrack->
   GetDefinition());
  }
  else { // This is primary
    Analysis::GetInstance()->SetBeam(
      aTrack->GetDefinition(),  // Getting info from Track
      aTrack->GetKineticEnergy()
    );
  }
  return result;
}
```

Le tracce secondarie mantengono memoria della traccia che le ha generate attraverso il metodo `GetParentID()` , si possono identificare le tracce primarie perchè hanno `parentID=0` .

3.2.3 SteppingAction

La simulazione procede attraverso `G4Steps` : la particella viene sottoposta a diversi processi fisici che modificano energia e momento della `G4Track` associata o creano tracce secondarie, il `G4Step` memorizza le informazioni che riguardandano questo *delta*. Uno step conosce il volume del detector e il materiale in cui si svolge: questa informazione è codificata dai `PreStepPoint` e `PostStepPoint` . Uno step non può mai avvenire a cavallo di due volumi diversi del detector.

La `SteppingAction` svolge la funzione del `SensitiveDetector` (cfr 2.2.3 a pagina 15): l'utente può fornire una funzione `UserSteppingAction(const G4Step * theStep )` che viene chiamata ad ogni step per registrare le informazioni necessarie. Il metodo del `SensitiveDetector` lavora tramite la funzione `ProcessHits` solo su specifici volumi attivi del detector, mentre la `SteppingAction` è molto più generica e flessibile, ma è necessario effettuare più controlli di tipo geometrico per capire in che regione del detector avviene lo step.

```
void SteppingAction::UserSteppingAction( const G4Step *
    theStep )
{
// Energia totale depositata nello step
G4double edep = theStep->GetTotalEnergyDeposit();
if(edep == 0.0) { return; }

// Vogliamo analizzare solo gli step avvenuti nel volume del
    calorimetro
const G4VTouchable* touchable = theStep->GetPreStepPoint()->
    GetTouchable();
G4int volCopyNum = touchable->GetVolume()->GetCopyNo();
if ( volCopyNum == 10 || volCopyNum == 11 ) //EM calo step
{
// Randomizzazione del punto di interazione tra PreStepPoint
    e PostStepPoint
G4double z1 = theStep->GetPreStepPoint()->GetPosition().z();
G4double z2 = theStep->GetPostStepPoint()->GetPosition().z();
G4double z  = z1 + G4UniformRand()*(z2 - z1);

// Salva informazioni con Analysis
Analysis::GetInstance()->AddEDepEM( edep, z, volCopyNum );
}}
```

In questo esercizio si vuole salvare il punto di interazione della particella all'interno del calorimetro e l'energia depositata. É necessario controllare il `copynumber` del volume in cui avviene lo step: questa informazione è recuperabile dal `PreStepPoint` [1] che fornisce un oggetto `G4VTouchable` contentente tutte le informazioni necessarie per identificare univocamente una regione del detector.

3.3 Risultati

In questo esercizio si è utilizzata la `PhysicsList` elettromagnetica standard (come negli esercizi precedenti 1.2.3 a pagina 8) e un `PrimaryGeneratorAction` che genera un fascio monoenergetico e di distribuzione gaussiana centrata nell'origine del piano xy. Si esegue una simulazione con 10000 π^+ ed e^- da 2 GeV.

3.3.1 Pioni

Si riporta l'output del run con 10000 π^+ da 2 GeV. In figura 3.1 nella pagina seguente si osservano 100 pioni che interagiscono con il calorimetro elettromagetico producendo gamma ed elettroni secondari (poco visibili perchè a basso range). Il

[1] Si noti che le informazioni geometriche si leggono sempre dal `PreStepPoint` per evitare problemi nel caso il `PostStepPoint` sia sulla superficie che divide due volumi (limite geometrico dello step).

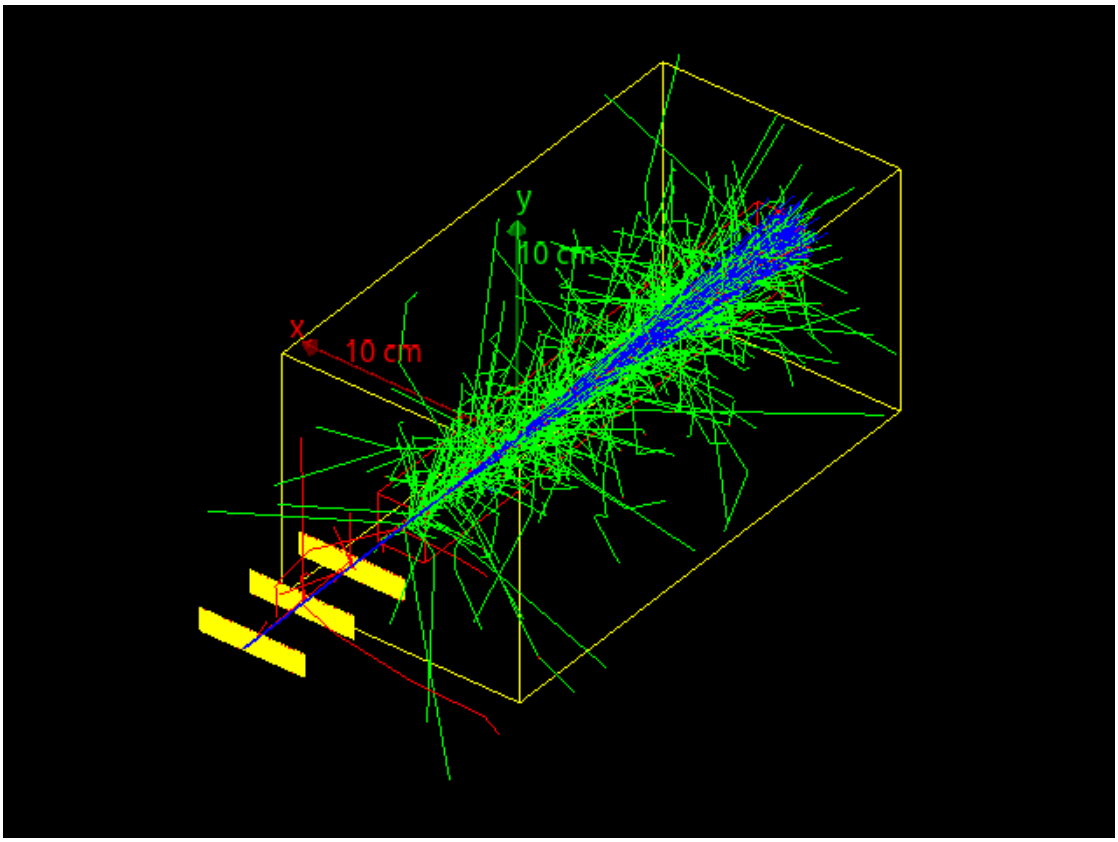

Figura 3.1: Fascio da 100 π^+ da 2 GeV

log della simulazione mostra infatti che in media vengono prodotti 362 elettroni e 63 gamma ad evento. La deposizione media di energia è bassa, concentrata nella regione centrale (vedi figura 3.2 nella pagina successiva) e distribuita lungo tutto il calorimetro (vedi figura 3.3 a pagina 29): la maggiorparte dei pioni sopravvive al calorimetro elettromagnetico e necessita di un calorimetro adronico per il completo assorbimento.

```
==================
Summary for run: 0
Beam of pi+ kinetic energy: 2 GeV
Event processed:          10000
Average number of gamma: 63.174
Average number of e-    : 362.789
Average number of e+    : 0.8272
Average energy deposition in EM calo: 272.353 MeV  (13.6 %)
Normalized energy in EM calo:          0.136176  RMS: 0.016129
Normalized energy in central crystal: 0.132969  RMS:
    0.0136327
Ratio of central crystal to total:    0.97645
==================
```

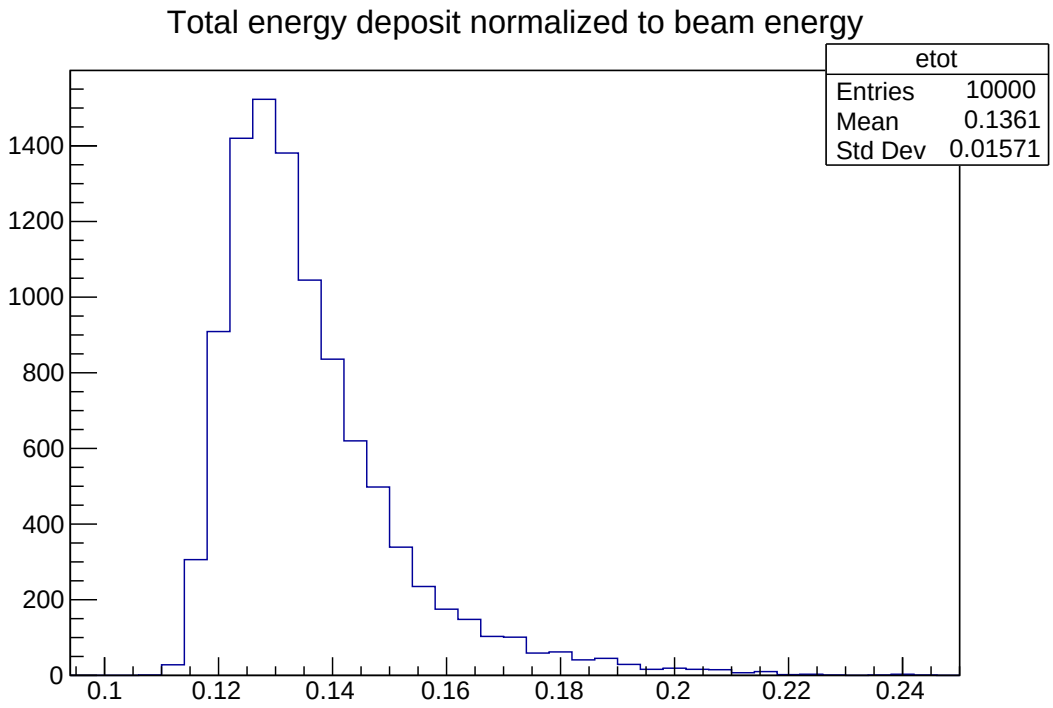

Figura 3.2: Energia totale depositata da π^+ da 2 GeV

3.3.2 Elettroni

Il risultato della simulazione di 1000 e^- da 2 GeV è molto diverso.

```
==================
Summary for run: 0
Beam of e- kinetic energy: 2 GeV
Event processed:          10000
Average number of gamma: 2003.56
Average number of e-    : 4337.74
Average number of e+    : 106.475
Average energy deposition in EM calo: 1.95236 GeV (97.6 %)
Normalized energy in EM calo:          0.976178  RMS:
    0.00704249
Normalized energy in central crystal: 0.77904  RMS: 0.0249717
Ratio of central crystal to total:    0.798051
==================
```

In figura 3.4 nella pagina successiva si mostra l'interazione di un singolo elettrone con il calorimetro: l'elettrone è convertito in una shower elettromagnetica di gamma, elettroni e positroni che viene asssorbita dal cristallo quasi completamente (97.6 % dell'energia iniziale) a causa della piccola lunghezza di radiazione del tungstato di piombo. In figura 3.6 a pagina 30 si mostra il profilo della deposi-

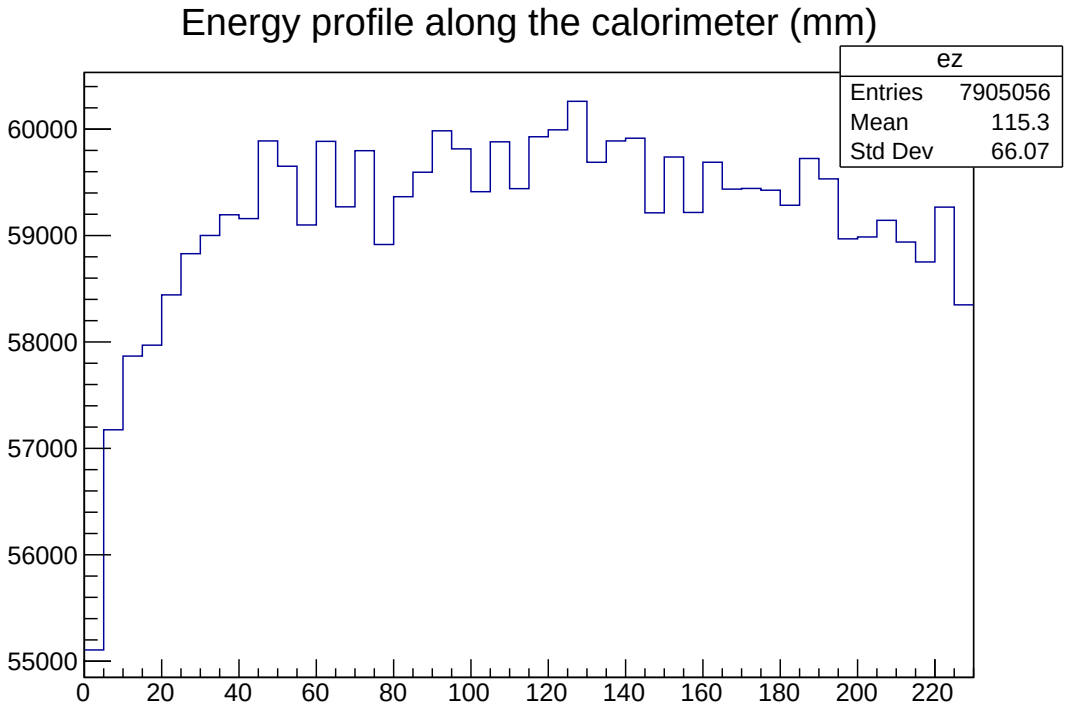

Figura 3.3: Energia depositata rispetto alla profondità del calorimetro per π^+ da 2 GeV

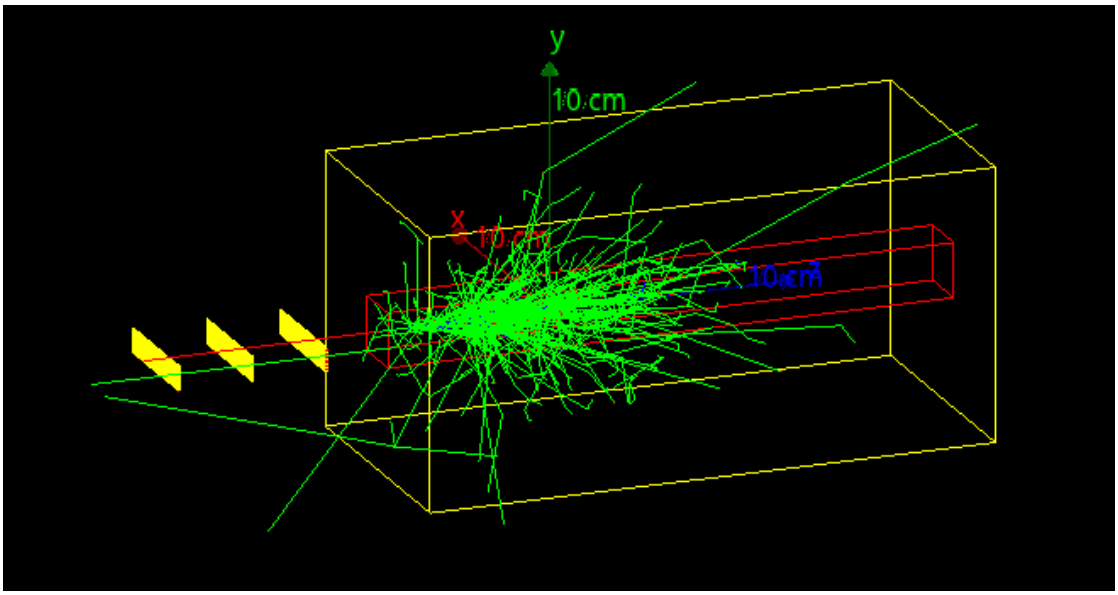

Figura 3.4: Singolo e^- da 2 GeV e shower elettromagnetica

zione di energia nel calorimetro: non è uniforme come per i pioni, ma presenta un picco a 45 mm di profondità.

In figura 3.7 a pagina 31 sono mostrate le tracce di elettroni e positroni nel dettaglio di una shower elettromagnetica. Per produrre questo output, nasconden-

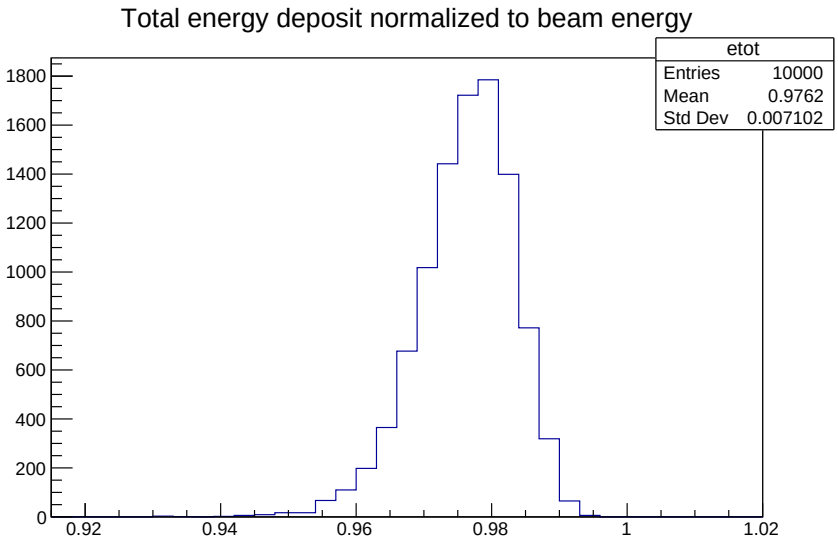

Figura 3.5: Energia totale depositata da e^- da 2 GeV

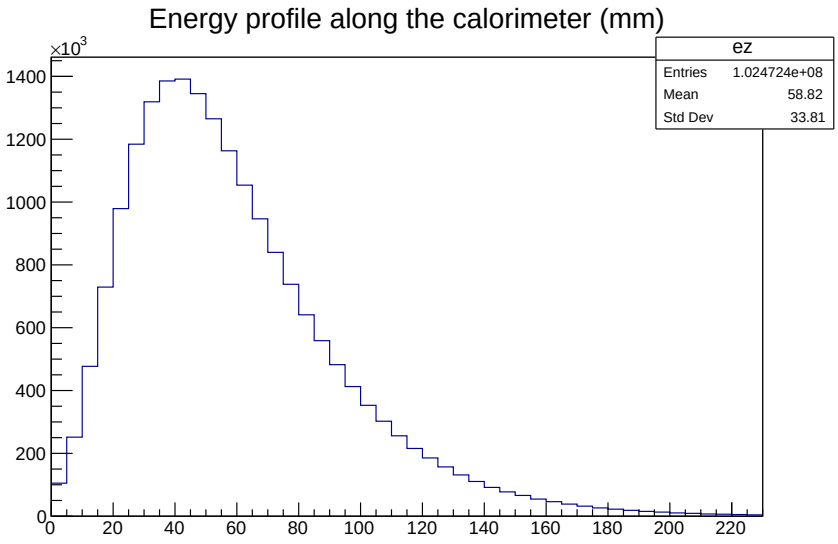

Figura 3.6: Energia depositata rispetto alla profondità del calorimetro per e^- da 2 GeV

do i gamma, sono stati utilizzati i segenti comandi macro per definire un filtro in visualizzazione.

```
/vis/filtering/trajectories/create/particleFilter
/vis/filtering/trajectories/particleFilter-0/add e-
/vis/filtering/trajectories/particleFilter-0/add e+
```

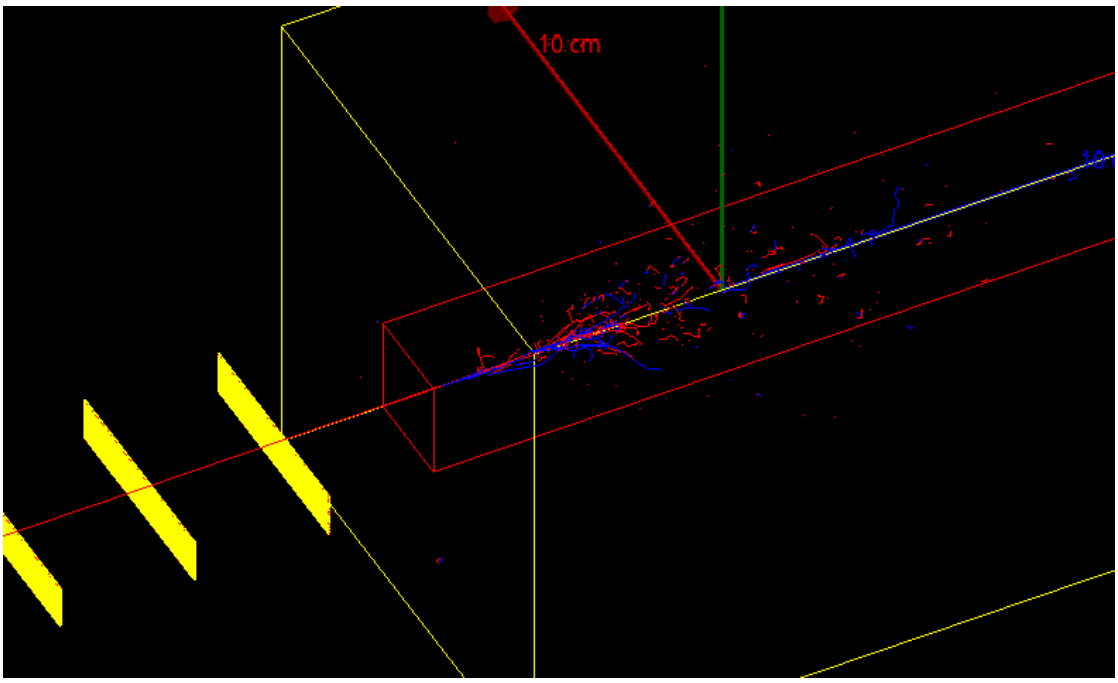

Figura 3.7: Dettaglio di elettroni (rosso) e positroni (blue) nella shower elettromagnetica

Capitolo 4

Esercizio 3b

4.1 Descrizione generale

In questo esercizio si ricostruisce il detector oggetto dell'esercizio 1 (telescopio di silicio, calorimetro elettromagnetico, calorimetro adronico di ferro e argon liquido) e si aggiunge un campo magnetico nel calorimetro adronico. Lo scopo è quello di studiare il decadimento del muone e la violazione di parità dell'interazione debole. A causa della violazione di parità l'elettrone prodotto dal decadimento tende ad essere emesso nella direzione opposta allo spin del muone, ma questo precede a causa della presenza del campo magnetico: il risultato è un'asimmetria nell'emissione dell'elettrone backward o forward. Per osservare questo effetto è necessario che il periodo di precessione dello spin sia confrontabile con la vita media del muone $\tau \simeq 2.6\,\mu s$. Essendo la frequenza di Larmor $\omega_l = eB/2mc$ si utilizza un campo magnetico di 3.5 mT che dà un periodo di precessione di 2.1 μs.

4.2 Componenti del programma

Si illustrano le modifiche al codice necessarie per aggiungere il campo magnetico e il processo di decadimento del muone.

4.2.1 DetectorConstruction

Per aggiungere un campo magnetico a una parte del detector è necessario instanziare un `G4FieldManager` che gestisce l'interazione delle particelle con il campo attraverso l'implementazione dell'equazione del moto e di un integratore.

```
// pure magnetic field
G4MagneticField* fMagneticField =
new G4UniformMagField(G4ThreeVector(3.5e-3*tesla, 0., 0.));
// equation of motion with spin
G4Mag_EqRhs* fEquation = new G4Mag_SpinEqRhs(fMagneticField);
// local field manager
```

```
G4FieldManager* fFieldManager = new G4FieldManager();
fFieldManager->SetDetectorField(fMagneticField );

// default stepper Runge Kutta 4th order
G4MagIntegratorStepper* fStepper = new G4ClassicalRK4(
    fEquation , 12); // spin needs 12 dof
// add chord finder
G4double fMinStep=1*mm;
G4ChordFinder* fChordFinder = new G4ChordFinder(
    fMagneticField, fMinStep,fStepper);
fFieldManager->SetChordFinder( fChordFinder );
```

Il `FieldManager` va poi associato al `LogicalVolume` in cui si vuole attivare il campo.

```
hadCaloLogic->SetFieldManager(GetLocalFieldManager(),true);
```

4.2.2 PhysicsList

In questo esercizio non si è utilizza la `PhysicsList` elettromagnetica standard ma si sono instanziate particelle e processi manualmente. E' interessante mostrare come viene attivato il decay dei muoni: prima di tutto in `ConstructParticle` va definita la `DecayTable`

```
// ********** PhysicisList::ConstructParticle() ********
G4MuonPlus::MuonPlusDefinition();
G4MuonMinus::MuonMinusDefinition();

G4DecayTable* MuonPlusDecayTable = new G4DecayTable();
MuonPlusDecayTable -> Insert(new G4MuonDecayChannelWithSpin("
    mu+",0.986));
MuonPlusDecayTable -> Insert(new
    G4MuonRadiativeDecayChannelWithSpin("mu+",0.014));
G4MuonPlus::MuonPlusDefinition() -> SetDecayTable(
    MuonPlusDecayTable);

G4DecayTable* MuonMinusDecayTable = new G4DecayTable();
MuonMinusDecayTable -> Insert(new G4MuonDecayChannelWithSpin(
    "mu-",0.986));
MuonMinusDecayTable -> Insert(new
    G4MuonRadiativeDecayChannelWithSpin("mu-",0.014));
G4MuonMinus::MuonMinusDefinition() -> SetDecayTable(
    MuonMinusDecayTable);
```

In seguito in `ConstructProcess` va attivato il processo di decadimento.

```
// ********** PhysicsList::ConstructProcess() **********
G4Decay* theDecayProcess = new G4DecayWithSpin();

G4ParticleDefinition* muMinus= G4MuonMinus::
    MuonMinusDefinition();
// Il ProcessManager va recuperato da ParticleDefinition
G4ProcessManager* muMinusManager = muMinus->GetProcessManager
    ();

// Il processo viene aggiunto al manager.
muMinusManager->AddProcess(theDecayProcess, 1, -1, 2);

// Lo stesso avviene per il muone+
```

Gli indici passati al `ProcessManager` corrispondono all'ordine in cui verranno eseguiti i processi che compongono uno step: il primo indice si riferisce ai processi da svolgere `AtRest` , il secondo `AlongStep` e il terzo `PostStep` . I primi due processi devono essere sempre `Transportation` e `Multiscattering` poichè limitano geometricamente la lunghezza dello step. Nel cado del decadimento viene attivato il decadimento da fermo con il primo indice e il decadimento in volo con il terzo.

Nella `PhysicsList` di questo esercizio vengono attivati manualmente tutti i processi di interesse per le varie particelle utilizzate. Ad esempio viene attivata la produzione di coppie per i pioni.

```
if( particleName == "mu+" || particleName == "mu-"    ) {
pmanager->AddProcess(new G4hMultipleScattering, -1, 1, 1);
pmanager->AddProcess(new G4hIonisation,         -1, 2, 2);
pmanager->AddProcess(new G4hBremsstrahlung,     -1, 3, 3);
pmanager->AddProcess(new G4hPairProduction,     -1, 4, 4);
}
```

4.2.3 Analysis

L'analisi dei dati si svolge come nell'esercizio precedente (vedi 3.2.1 a pagina 23) utilizzando un singleton della classe Analysis. Si vogliono memorizzare informazioni sul decadimento del muone: per farlo è necessario intercettare le nuove tracce nella `StackingAction` (cfr esercizio3a 3.2.2 a pagina 24) controllando se provengono da un processo di decadimento.

```
// ********** StackingAction::ClassifyNewTrack() ********
if ( aTrack->GetParentID() > 0 )//This is a secondary
{
  Analysis::GetInstance()->AddSecondary(1);
  // Controllo sul processo che ha generato la traccia
```

```
6   if ( aTrack->GetCreatorProcess()->GetProcessType()==fDecay
     ) {
7     // Salva informazioni sul decadimento
8     analysis->AddTrack(aTrack);
9   }
10 }
```

Nella classe `Analysis` è possibile estrarre la posizione e il tempo di decadimento.

```
1  // *********** Analysis.cc ***************
2  void Analysis::AddTrack( const G4Track * aTrack )
3  {
4    // Controllo che le tracce siano elettroni
5    if (aTrack->GetDefinition()->GetPDGEncoding()!=11) return;
6      const G4ThreeVector & pos = aTrack->GetPosition();
7      // Tempo globale della simulazione (sistema del lab)
8      G4double time = aTrack->GetGlobalTime();
9      histos[fDecayPosZ]->Fill(pos.z()/m);
10     histos[fDecayTime]->Fill(time/microsecond);
11   }
12 }
```

4.3 Risultati

Si è eseguita la simulazione di 100000 μ^- da 1 GeV. Il plot 4.1 nella pagina seguente mostra la distribuzione dei tempo di decadimento del muone: dal fit esponenziale si ricava la vita media $\tau = 2.168 \pm 0.007\,\mu m$. Osservando la distribuzione del tempo di decadimento dei muoni con elettrone emesso in avanti o indietro (figure 4.2 nella pagina successiva e 4.3 a pagina 37) si nota che si ha una variazione periodica che corrisponde alla precessione dello spin a causa del campo magnetico: gli elettroni tendono ad essere emessi nella direzione opposta allo spin del muone.

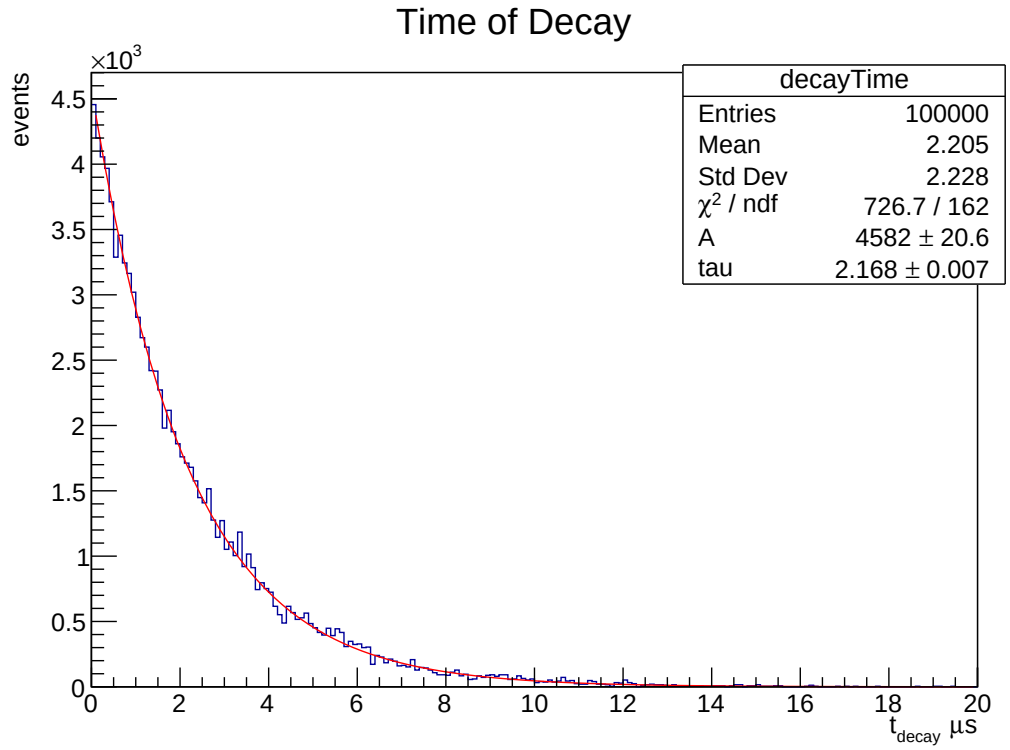

Figura 4.1: Distribuzione del tempo di decadimento del muone

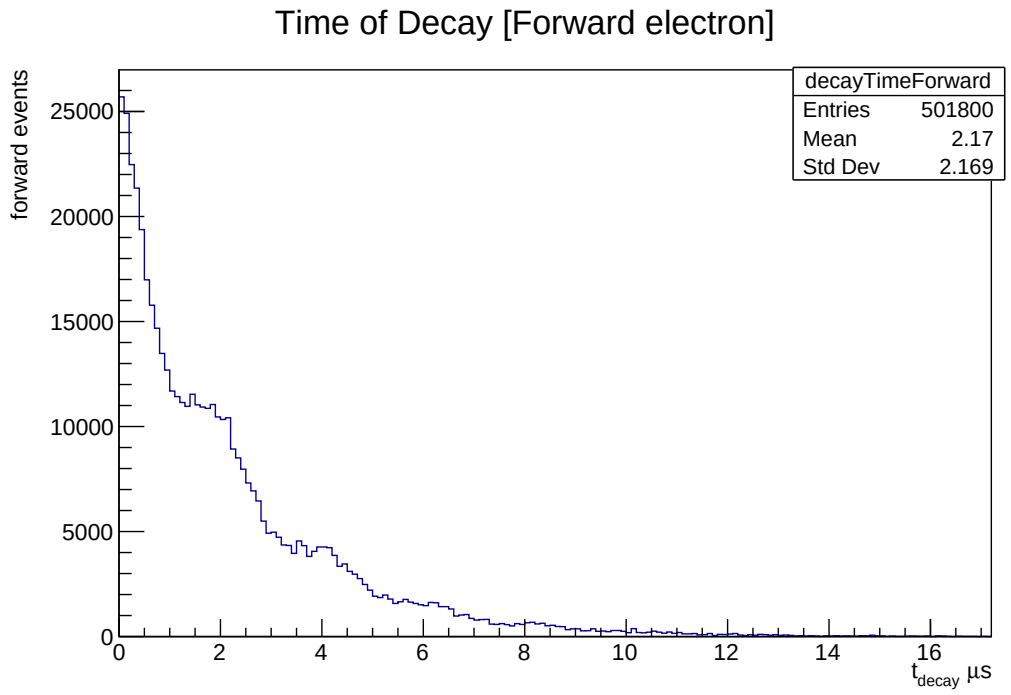

Figura 4.2: Distribuzione del tempo di decadimento del muone con elettrone emesso in avanti

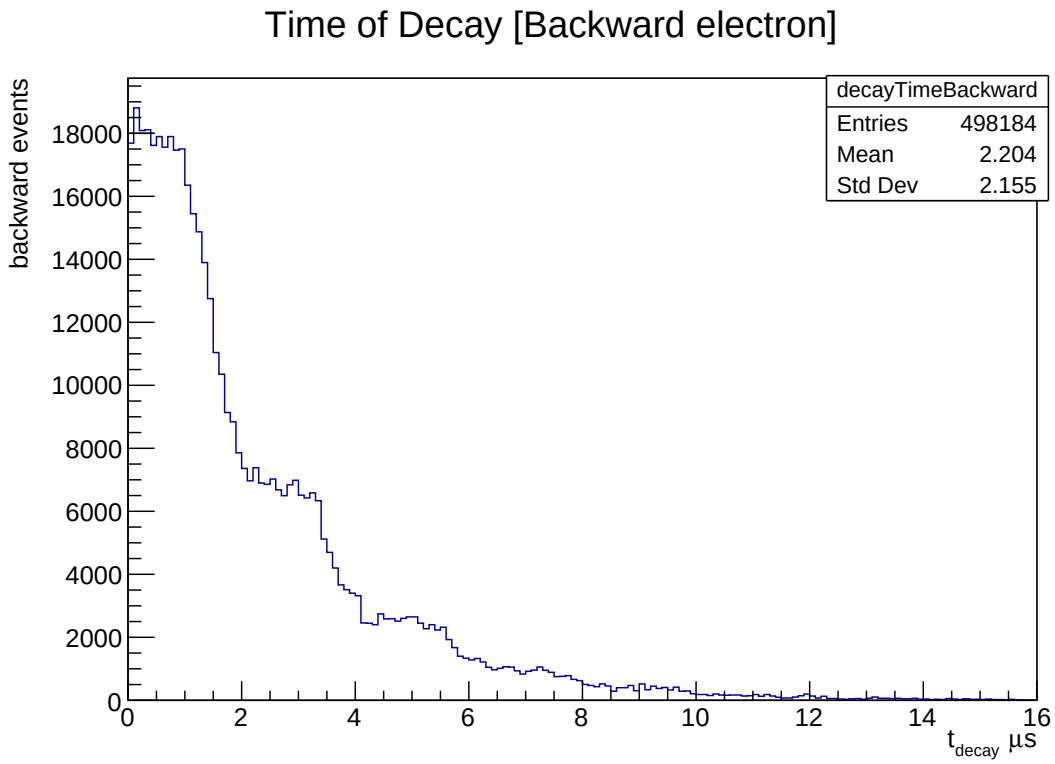

Figura 4.3: Distribuzione del tempo di decadimento del muone con elettrone emesso indietro

Capitolo 5

Esercizio 4

5.1 Descrizione generale

In questo esercizio si modifica la simulazione precedente aggiungendo un `Sensitive Detector` agli strati di argon liquido del calorimetro elettromagnetico per memorizzare più facilmente l'energia depositata in ogni livello rispetto all'utilizzo generico della `SteppingAction` (vedi anche il confronto tra i due metodi nella sezione 3.2.3 a pagina 25).

Inoltre non viene utilizzata una `PhysicsList` definita manualmente dall'utente come è stato fatto finora, ma nel main della simulazione viene instanziata una `PhysicsList` predefinta di Geant4 la `QGSP_BERT`.

```
// ********** main() ************
G4VUserPhysicsList* physics = new QGSP_BERT();
runManager->SetUserInitialization(physics);
```

Questa `PhysicsList` applica il modello a stringhe di quark e gluon per le interazioni ad alta energia ($>$ 10 GeV) di protoni, neutroni, pioni, kaoni e nuclei. Le interazioni ad alta energia creano nuclei eccitati, la cui diseccitazione è simulata con il modello precompound. Per le particelle primarie di energia inferiore ai 10 GeV l'interazione è modellata con la *Bertini cascade* che in genere produce più neutroni e protoni secondari del modello LEP, con un accordo maggiore ai dati sperimentali. [1].

5.2 Esercizio 4a

Le uniche differenze rispetto alla simulazione dell'esercizio 3 sono quelle descritte nella sezione generale. Nel `SensitiveDetector` si utilizzano gli `Hit` ma non si salvano in una `HitCollection` , si utilizza ancora direttamente la classe `Analysis` per raccogliere i dati.

[1] Pagina della documentazione ufficiale

Si riporta di seguito il risultato della simulazione di un protone da 1 GeV. Ad esempio si mostrano i processi di ionizzazione e produzione di coppie: per ogni processi possono esserci diversi modelli che agiscono su scale di energie different.

```
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


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 protone

```
---------------------------------------------------
Hadronic Processes for proton

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

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

e i parametri relativi al processo di Pre-Compound

```
#=========================================================
#     Pre-compound/De-excitation Physics Parameters
```

```
#=========================================================
Type of pre-compound inverse x-section                3
Pre-compound model active                             1
Pre-compound low energy (MeV)                         0.1
Type of de-excitation inverse x-section               3
Type of de-excitation factory              Evaporation
Number of de-excitation channels                      8
Min excitation energy (keV)                           0.01
Min energy per nucleon for multifragmentation (MeV)   1e+05
Level density (1/MeV)                                 0.1
Time limit for long lived isomeres (ns)               1e+12
Internal e- conversion flag                           1
Store e- internal conversion data                     0
Electron internal conversion ID                       2
Correlated gamma emission flag                        0
Max 2J for sampling of angular correlations           10
========================================================================
```

Per ogni regione del detector vengono poi calcolati cut in energia e range.

```
Material : G4_Fe
Range cuts        :  gamma  700 um     e-  700 um     e+  700
    um  proton 700 um
Energy thresholds :  gamma  17.2183 keV    e-  951.321 keV
    e+  901.528 keV proton 70 keV
Region(s) which use this couple :
DefaultRegionForTheWorld
Index : 4     used in the geometry : Yes
Material : G4_lAr
Range cuts        :  gamma  700 um     e-  700 um     e+  700
    um  proton 700 um
Energy thresholds :  gamma  5.20596 keV    e-  272.577 keV
    e+  267.137 keV proton 70 keV
Region(s) which use this couple :
DefaultRegionForTheWorld
```

Infine si mostrano i dati di tutti gli `Hit` e le statistiche: da notare che la particella primaria non arriva agli strati attivi del calorimetro adronico che registra invece protoni, neutroni, elettroni e gamma secondari. Inoltre si notano gli hit generati da nuclei provenienti dalla disintegrazione. Dagli hit possiamo dedurra la massima profondità di interazione che è circa l'11-esimo strato di argon per protoni da 1 GeV (33 cm).

```
Layer: 0 (volume CopyNo: 1001) Edep=4.86615 MeV isPrimary? No
    (name=proton)
```

```
Layer: 1 (volume CopyNo: 1002) Edep=5.41812 MeV isPrimary? No (name=proton)
[...]
Layer: 1 (volume CopyNo: 1002) Edep=0 eV  isPrimary? No (name=neutron)
Layer: 10 (volume CopyNo: 1011) Edep=5.62426 keV isPrimary? No (name=neutron)
[...]
Layer: 1 (volume CopyNo: 1002) Edep=93.9963 keV isPrimary? No (name=e-)
Layer: 1 (volume CopyNo: 1002) Edep=140.492 keV isPrimary? No (name=e-)
Layer: 1 (volume CopyNo: 1002) Edep=20.0736 keV isPrimary? No (name=e-)
[...]
Layer: 0 (volume CopyNo: 1001) Edep=0 eV  isPrimary? No (name=gamma)
Layer: 1 (volume CopyNo: 1002) Edep=0 eV  isPrimary? No (name=gamma)
Layer: 2 (volume CopyNo: 1003) Edep=0 eV  isPrimary? No (name=gamma)
[...]
Layer: 2 (volume CopyNo: 1003) Edep=197.669 keV isPrimary? No (name=Ar40)
Layer: 4 (volume CopyNo: 1005) Edep=126.066 keV isPrimary? No (name=Ar40)
Layer: 1 (volume CopyNo: 1002) Edep=363.59 keV isPrimary? No (name=K39)

Event: 0 Energy in EM calo: 269.837 MeV Secondaries: 239
User=0.06s Real=0.07s Sys=0s
=================
Summary for run: 0
Event processed: 1
Average number of secondaries: 239
Average energy in EM calo: 269.837 MeV
Average energy in Had calo: 49.5982 MeV
=================
```

5.3 Esercizio 4b

In questa parte dell'esercizio 4 si disattiva il calorimetro elettromagnetico e si utilizza la `StackingAction` per contare i neutroni e i gamma sopra una soglia di energia (30 MeV). Inoltre si vuole disabilitare la simulazione delle tracce gamma secondarie (dopo averle conteggiate).

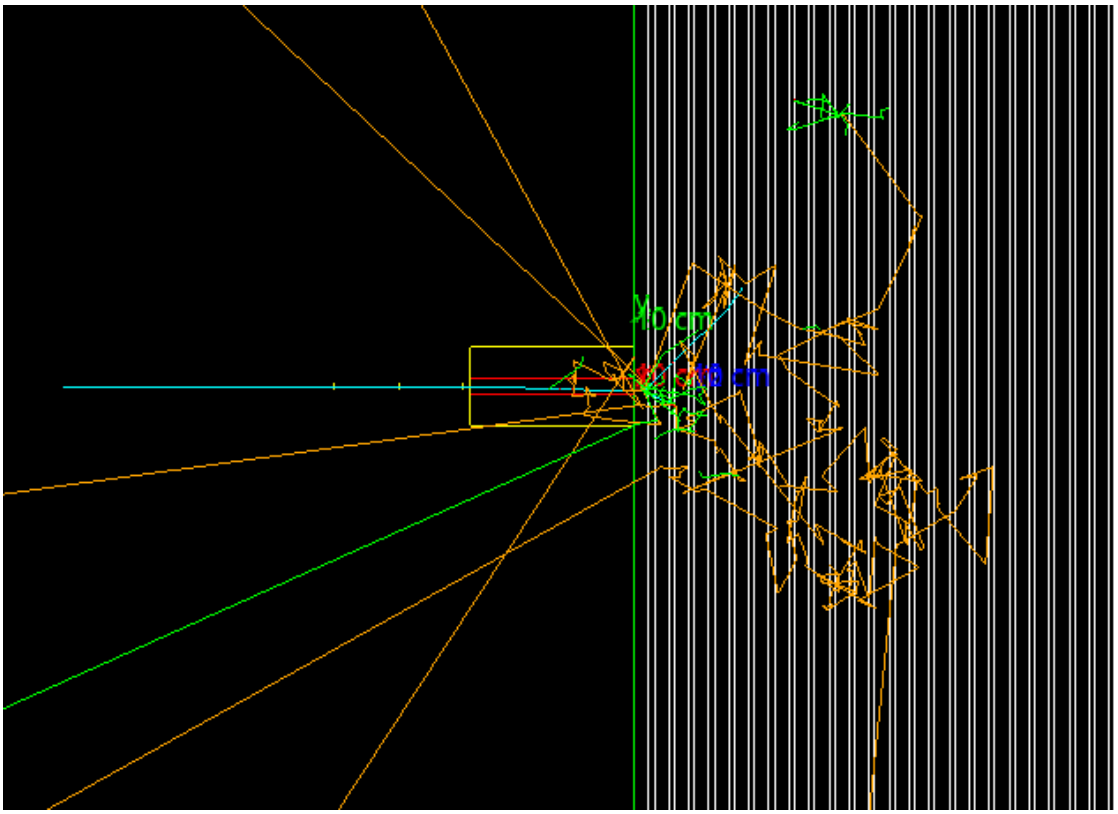

Figura 5.1: Interazione di un protone da 1 GeV con calorimetro elettromagnetico e adronico (in azzurro il protone, in arancione i neutroni e in verde i gamma)

```
// ***** StackingAction::ClassifyNewTrack() *****
// Tipo di particella della traccia
G4ParticleDefinition * particleType = aTrack->GetDefinition()
    ;

G4double thresh = 30*MeV;

// Si conteggiano gamma e neutroni sopra soglia
if ( particleType == G4Gamma::GammaDefinition() ){
  if (aTrack->GetKineticEnergy() > thresh){
    analysis->AddGammas(1);
  }
}
if ( particleType == G4Neutron::NeutronDefinition() ){
  if (aTrack->GetKineticEnergy() > thresh){
    analysis->AddNeutrons(1);
  }
}

// Se la particella e' un gamma cancellare la track
```

```
// impostando il risultato della unzione a fKill
if ( particleType == G4Gamma::GammaDefinition() ){
  result = fKill;
}
```

Si è utilizzato un fascio di π^- e uno di neutroni di diverse energie per testare il conteggio della particelle: il numero di gamma e neutroni secondari sopra soglia è simile per i due tipi di fascio.

```
Run Summary  PIi- 1 GeV
Number of events processed : 1000
User=28.96s Real=29.3s Sys=0.26s
=================
Event processed: 1000
Average number of secondaries: 252
Average energy in Had calo: 42.1412 MeV
Average number of gammas: 0
Average number of neutrons: 3
=================

Run Summary Pi- 10 GeV
Number of events processed : 1000
User=81.52s Real=81.97s Sys=0.31s
=================
Event processed: 1000
Average number of secondaries: 1628
Average energy in Had calo: 272.224 MeV
Average number of gammas: 7
Average number of neutrons: 28
=================

Run Summary Pi- 50 GeV
Number of events processed : 1000
=================
Event processed: 1000
Average number of secondaries: 6097
Average energy in Had calo: 1.05894 GeV
Average number of gammas: 34
Average number of neutrons: 103
=================


**********************************************
Run Summary  Neutron 1 GeV
Number of events processed : 1000
User=14.39s Real=14.52s Sys=0.12s
=================
Event processed: 1000
```

```
Average number of secondaries: 264
Average energy in Had calo: 34.1541 MeV
Average number of gammas: 0
Average number of neutrons: 6
==================

Run Summary Neutron 10 GeV
Number of events processed : 1000
User=101.93s Real=102.46s Sys=0.04s
==================
Event processed: 1000
Average number of secondaries: 1842
Average energy in Had calo: 294.614 MeV
Average number of gammas: 7
Average number of neutrons: 36
==================

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
==================
```

La simulazione non mostra tracce di gamma secondari poichè sono state eleminate dalla `StackingAction` (figura 5.2 nella pagina seguente).

5.4 Esercizio 4c

In questo esercizio si modifica la simulazione precedente aggiungendo il salvataggio degli `Hit` nel `SensitiveDetector` . Poichè in un calorimetro la generazione di `Hit` può essere inefficiente si utilizza una mappa nel `SensitiveDetector` per memorizzare solo un `Hit` per piano di argon e si accumula l'energia depositata. Nell'header del `SensitiveDetector` è necessario definire:

```
// Helper mapping layer number with hit
typedef std::map<G4int,HadCaloHit*> hitMap_t;
hitMap_t hitMap;
```

Nella funzione `ProcessHits` del `SensitiveDetector` si recupera il livello della hit e l'energia depositata e si accumula nella mappa.

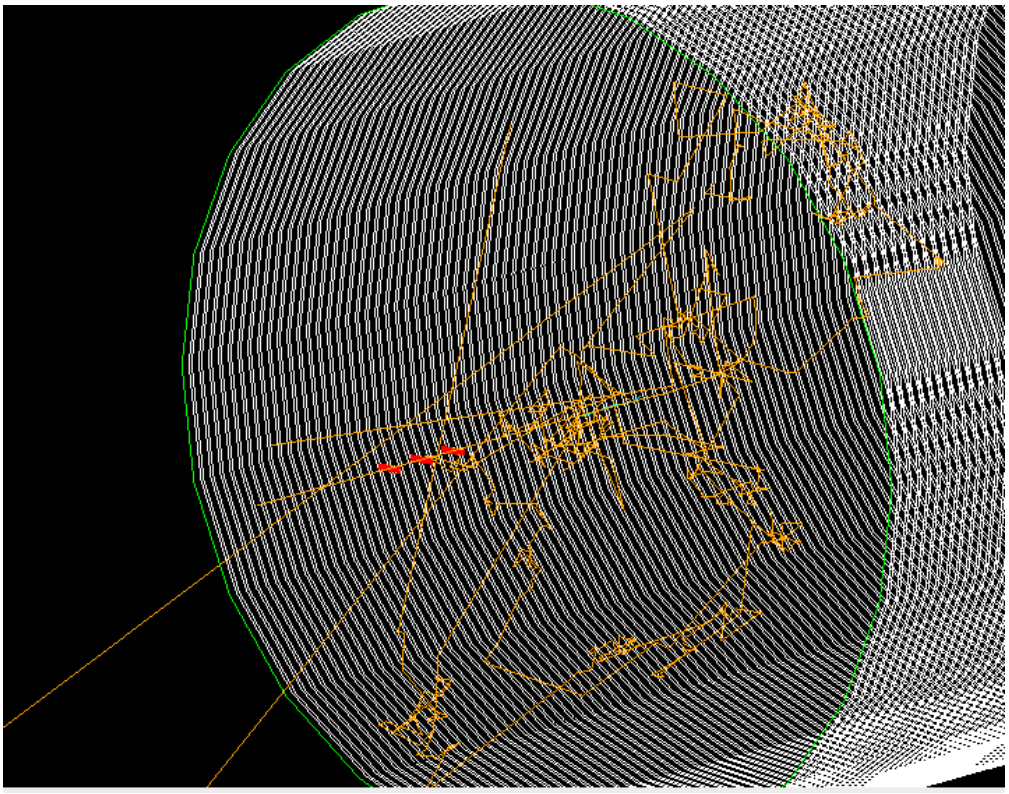

Figura 5.2: Neutrone da 1 GeV su calorimetro adronico (i neutroni sono arancioni, i protoni azzurro)

```
G4bool HadCaloSensitiveDetector::ProcessHits(G4Step *step,
    G4TouchableHistory *)
{
// Recupera touchable per il copynumber del volume
G4TouchableHandle touchable = step->GetPreStepPoint()->
    GetTouchableHandle();
G4int copyNo = touchable->GetVolume(0)->GetCopyNo();
//Hadronic layers have number from 1001 to 1080. The index is
     from 0 to 79:
G4int layerIndex = copyNo-1001;
//We get now the energy deposited by this step
G4double edep = step->GetTotalEnergyDeposit();

// Si recupera l'hit per questo piano o ne si crea uno nuovo
hitMap_t::iterator it = hitMap.find(layerIndex);
HadCaloHit* aHit = 0;
if ( it != hitMap.end()) {
  aHit = it->second;
} else {
  // Crea un nuovo hit per il livello
  aHit = new HadCaloHit(layerIndex);
```

```
19   hitMap.insert( std::make_pair(layerIndex,aHit));
20   hitCollection->insert(aHit);
21 }
22 // Accumula l'energia
23 aHit->AddEdep( edep );
```

Le informazioni raccolte vengono mostrate alla fine di ogni evento.

```
1 void HadCaloSensitiveDetector::EndOfEvent(G4HCofThisEvent*)
2 {
3 // Metodo di HitCollection -> stampa tutte le hit
4 hitCollection->PrintAllHits();
5 }
```

In `Analysis.cc` si recupera la collezione degli hit alla fine di ogni evento per calcolare la media di energia depositata per ogni run: è necessario utilizzare l'id della collezione e recuperare l'oggetto `G4HCofThisEvent` , la collezione degli eventi dell'evento corrente creata dal `SensitiveDetector` .

```
1 G4SDManager* SDman = G4SDManager::GetSDMpointer();
2 G4int hitCollID = SDman->GetCollectionID("
    HadCaloHitCollection");
3 G4HCofThisEvent* hitsCollections = anEvent->GetHCofThisEvent
    ();
4 HadCaloHitCollection* hits = 0;
5 if ( hitsCollections )
6 {
7   hits = static_cast<HadCaloHitCollection*>(hitsCollections->
    GetHC(hitCollID));
8 }
9 // ... analysis ...
```

Si presenta il risultato di un run di 100 π^+ da 2 GeV e di uno di neutroni della stessa energia: per ogni evento viene mostrata l'energia depositata per livello, infine l'energia media sul run.

```
1 ### Run 0 starts.
2 Starting Run: 0
3 Starting Event: 0
4 Energy Deposited in layer 0 is 1.40622 MeV
5 Energy Deposited in layer 1 is 2.07927 MeV
6 Energy Deposited in layer 2 is 1.39722 MeV
7 Energy Deposited in layer 3 is 2.20579 MeV
8 Energy Deposited in layer 4 is 3.71753 MeV
9 Energy Deposited in layer 5 is 24.4079 MeV
10 Energy Deposited in layer 6 is 5.50504 MeV
```

```
Energy Deposited in layer 7 is 4.68658 MeV
Energy Deposited in layer 8 is 2.25672 MeV
Energy Deposited in layer 9 is 2.2017 MeV
Energy Deposited in layer 10 is 2.72695 MeV
Energy Deposited in layer 11 is 7.34609 MeV
Energy Deposited in layer 13 is 2.73438 MeV
Energy Deposited in layer 19 is 4.414 eV
Energy Deposited in layer 24 is 7.25014 MeV
Energy Deposited in layer 23 is 5.19309 MeV
[...]
==================
Summary for run: 0
Event processed: 100
Average number of secondaries: 1889
Average energy in Had calo: 110.833 MeV

Average energy in Layer 0: 5.57862 MeV
Average energy in Layer 1: 5.9307 MeV
Average energy in Layer 2: 7.50282 MeV
Average energy in Layer 3: 6.63067 MeV
Average energy in Layer 4: 7.0253 MeV
Average energy in Layer 5: 6.00469 MeV
Average energy in Layer 6: 7.05424 MeV
Average energy in Layer 7: 6.28525 MeV
Average energy in Layer 8: 7.78059 MeV
Average energy in Layer 9: 7.22473 MeV
Average energy in Layer 10: 4.24887 MeV
Average energy in Layer 11: 2.84577 MeV
[...]
```

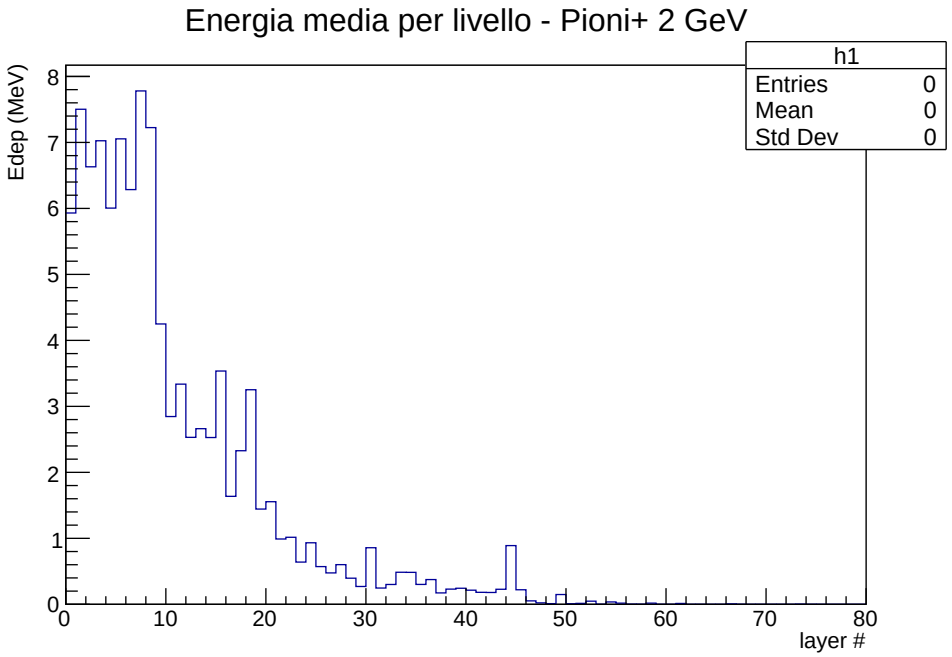

Figura 5.3: Energia media depositata da π^+ da 2 GeV per strato del calorimetro

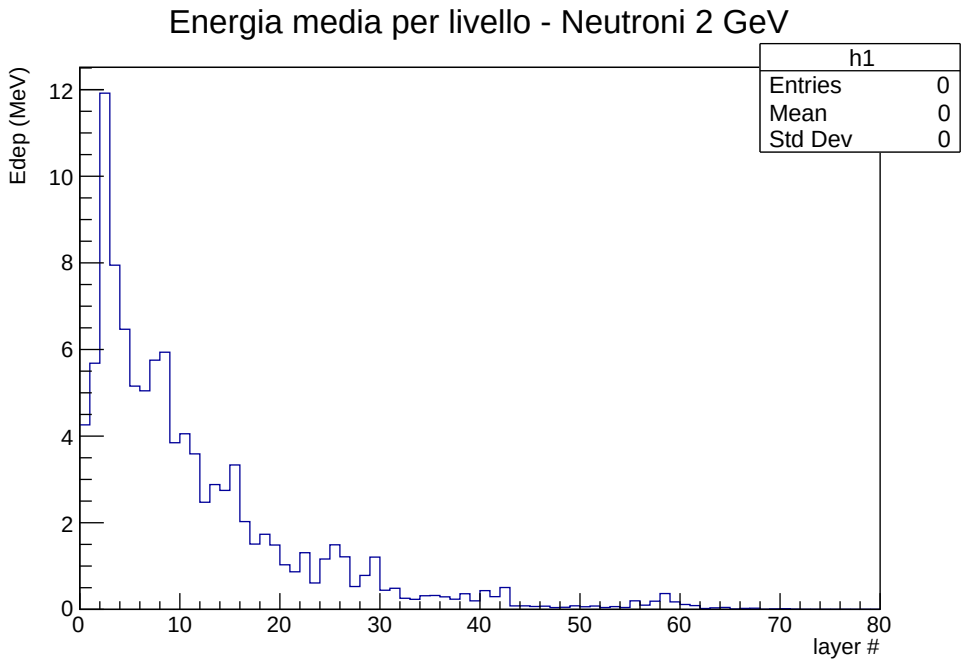

Figura 5.4: Energia media depositata da neutroni da 2 GeV per strato del calorimetro. I neutroni depositano più energia dei pioni nei primi strati del detector.

Capitolo 6

Esercizio 6

6.1 Descrizione generale

In questo esercizio si studia l'efficienza di rivelazione di neutroni veloci con un detector 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 (CH_2) è un materiale adatto. Verranno poi testati diversi materiali: rame, oro, alluminio, carbonio.

L'efficienza del detector viene quantificata con la probabilità di conversione dei neutroni e l'energia depositata del gas dalle particelle secondarie.

6.2 Componenti del programma

6.2.1 DetectorConstruction

Lo strato per la conversione dei neutroni è costruito con un parallelepipedo di 20 cm * 20 cm * 100 μm. Il materiale, polietilene (CH_2), viene costruito manualmente con il seguente codice, recuperando dal `G4NistManager` i singoli atomi e combinandoli nelle giuste proporzione e con la giusta densità in un materiale.

```
G4NistManager* man = G4NistManager::Instance();

G4int hydrogenZ = 1;
G4int carbonZ   = 6;

G4Element* H = man->FindOrBuildElement(hydrogenZ);
if(H) G4cout << "Hydrogen correctly retrieved" << G4endl;
G4Element* C = man->FindOrBuildElement(carbonZ);
if(C) G4cout << "Carbon correctly retrieved" << G4endl;

G4cout << "Building PE Material" << G4endl;

```

```
13 G4double PE_density = 0.935* g/cm3; // mean PE density
14
15 G4int PE_comp_Atoms = 2;
16 G4int n_H_Atoms = 2;
17 G4int n_C_Atoms = 1;
18
19 // Build PE material
20 PE_Mat = new G4Material("PEMat", PE_density, PE_comp_Atoms);
21
22 PE_Mat->AddElement(H,n_H_Atoms);
23 PE_Mat->AddElement(C,n_C_Atoms);
```

Il rivelatore a gas è costituito da una camera di 20 cm * 20 cm * 6 mm e riempito di Argon gassoso ad alta densità a cui si collega un `SensitiveDetector` .

```
1 G4int ArgonZ = 18;
2 G4Element* Ar = man->FindOrBuildElement(ArgonZ);
3
4 if(Ar) G4cout << "Argon correctly retrieved" << G4endl;
5
6 G4cout << "Building High density germanium Material" <<
    G4endl;
7
8 G4double argon_density = 0.001784* g/cm3;
9
10 G4int Ar_comp_Atoms = 1;
11
12 Ar_Mat = new G4Material("Ar_Mat",argon_density,Ar_comp_Atoms,
    kStateGas); ///You have to specify that it is a GAS!!!!);
13
14 Ar_Mat->AddElement(Ar,Ar_comp_Atoms);
```

6.2.2 DetectorMessenger

Come nell'esercizio 2 (vedi 2.2.2 a pagina 14) si utilizza un `DetectorMessenger` per modificare le caratteristiche del detector attraverso i comandi macro. In questo esercizio si vuole cambiare il materiale di cui è costituito il catodo. Si è implementata una funziona che cambia il materiale del catodo (utilizzando un material standard `Nist`) e aggiorna il detector.

```
1 void DetectorConstruction::ChangeCathodeMaterial(G4String
    new_material){
2
3   G4NistManager* man = G4NistManager::Instance();
```

```
  G4Material * mat = man->FindOrBuildMaterial("G4_Galactic");
  if(mat && logicPEConv) {
    // Set the material in the LogicVolume
    logicPEConv->SetMaterial(mat);
    //Update the geometry
    this->UpdateGeometry();
    G4cout << "Cathode material changed to: " << new_material
    << G4endl;
  }else{
    G4cout << "Failed to change cathode material!" << G4endl;
  }
}
```

Nella classe `DetectorMessenger` si implementa il comando.

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

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

6.2.3 SensitiveDetector

Nel `SensitiveDetector` si recuperano informazioni sulle `Track` che attraversano il rivelatore a gas e si creano degli oggetti `TrackParticle` che contengono tutte le informazioni interessanti (energia depositata, id traccia madre, tipo di particella, posizione Z di partenza della traccia). Viene creata una mappa di `TrackParticle` , `typedef std::map<G4int,TrackParticle*> trackMap_t;` per poter collegare ogni particella con le informazione della propria madre in seguito.

```
G4Track* thistrack   = step->GetTrack();

G4double trackenergy =thistrack->GetDynamicParticle()->
    GetKineticEnergy();
// Si usa l'Id della traccia come chiave per la mappa
G4int    thistrackID = thistrack->GetTrackID();
G4String trackname = thistrack->GetDefinition()->
    GetParticleName();
// Id della traccia madre
G4int    parentID    = thistrack->GetParentID();
// PDG id della particella
```

```
10 G4int trackPDGencod = thistrack->GetDefinition()->
       GetPDGEncoding();
11 // Posizione di partenza della traccia
12 G4double zstartpos = (thistrack->GetVertexPosition()).getZ();
13 G4double starttime = thistrack->GetGlobalTime();
```

6.2.4 Salvataggio dati

I dati di ogni evento vengono passati dall'`EventAction` alla classe `RootSaver`. Come nei precedenti esercizi, questa classe gestisce la creazione di un `TTree` per il salvataggio delle informazioni richieste.

La classe `EventAction` recupara la `trackMap` dal `SensitiveDetector` e la passa al `RootSaver`.

```
1 // ******* EventAction::EndOfEventAction *******
2 G4String sdname = "/myDet/ArCO2";
3 // Si recupera il s.d. attraverso l'id.
4 SensitiveDetector* sensitive = this->GetSensitiveDetector(
      sdname);
5 trackMap_t trackMap = sensitive->GetTrackMap();
6
7 rootSaver->AddEvent(trackMap, anEvent->GetEventID());
```

In `RootSaver` la funzione `GetParentProperties(trackMap)` si occupa di costruire una mappa di oggetti `TParentParticle`, analoghi ai `TParticle` ma contenenti tutte le informazioni relative alla particella madre, elaborando la `trackMap`

Per ogni evento viene creata un'entry nel `TTree` contente informazioni su ogni traccia secondaria (vengono ignorate le tracce dei neutroni primari che non sono stati convertiti convertiti).

```
1 rootTree->Branch( "ntracks" , &Tot_Tracks, "ntracks/I" );
2 rootTree->Branch( "id", PartID , "id[ntracks]/I");
3 rootTree->Branch( "mum", Part_Moth_ID  , "mum[ntracks]/I");
4 rootTree->Branch( "type", PartType , "type[ntracks]/I");
5 rootTree->Branch( "mtype",  MothPartType  , "mtype[ntracks]/I
      ");
6 rootTree->Branch( "edep", Part_EnDep  , "edep[ntracks]/F");
7 rootTree->Branch( "medep",  MothPart_EnDep , "medep[ntracks]/
      F");
8 rootTree->Branch( "zp", Part_zStart  , "zp[ntracks]/F");
9 rootTree->Branch( "mzp", MothPart_zStart  , "mzp[ntracks]/F")
      ;
10 rootTree->Branch( "t", Part_tStart  , "t[ntracks]/F");
11 rootTree->Branch( "mt", MothPart_tStart  , "mt[ntracks]/F");
12 rootTree->Branch("evid",&Event_ID,"evid/I");
```

6.2.5 PhysicsList

In questo esercizio si utilizza la `PhysicsList` `QGSP_BERT_HP` analoga a quella utilizzata nell'esercizio 4 (vedi 5.1 a pagina 38): questa `PhysicsList` è simile alla `QGSP_BERT` ma utilizza in aggiunta il pacchetto `NeutronHP` per il trasporto ad alta precisione dei neutroni sotto i 20 GeV fino alla terminalizzazione.

6.3 Risultati

6.3.1 Catodo in polietilene

In primo luogo si vuole caratterizzare il detector con catodo in polietilene in efficienza di conversione dei neutroni ed efficienza di rivelazione dell'energia delle tracce secondarie. Si sono effettuati diversi run da $1 \cdot 10^6$ neutroni di energia compresa tra i 500 KeV e i 20 MeV e si sono estratti questi valori:

- Efficienza di rivelazione come numero di eventi con tracce secondarie su numero di eventi con neutrone non interagente. (Figura 6.2 a pagina 55)
- Energia depositata percentuale dalle tracce secondarie nel rivelatore a gas. (Figura 6.3 a pagina 55)

L'efficienza di conversione del polietilene ha un massimo per neutroni di energia 4 MeV (0.09 %), mentre la percentuale di energia depositata nel gas diminuisce con l'energia iniziale dei neutroni e quindi con l'energia delle tracce secondarie (come noto dalla legge di Bethe-Block).

Si è analizzato inoltre il numero medio di tracce secondarie che raggiungono il rivelatore a gas 6.4 a pagina 56: esso raggiunge un massimo di ~ 15 tracce e si mantiene circa costante all'aumentare dell'energia: infatti il neutrone interagisce elasticamente con un nucleo di idrogeno, un protone, che ionizza elettroni nel gas, il cui numero non varia molto con l'energia del protone.

Dalle figure 6.5 a pagina 56 e 6.6 a pagina 57 si osserva che in media viene prodotto sempre un solo protone e che il numero di tracce secondarie corrisponde al numero di elettroni. L'analisi completa delle tracce ha evidenziato in alcuni eventi la presenza di neutroni secondari e nuclei di argon, calcio e carbonio che hanno subito un processo di scattering elastico con il neutrone.

In figura 6.7 a pagina 57 si mostra la distribuzione della posizione di partenza delle tracce di protoni provenienti dal catodo con un fascio primario di neutroni a 4 MeV su polietilene: si nota che è più probabile che raggiungano il detector i protoni prodotti più in profondità nel catodo, mentre quelli prodotti nei primi μm di materiale vengono riassorbiti prima di poter attraversare lo strato polietilene.

Infine in figura 6.8 a pagina 58 si mostra la distribuzione della posizione di partenza delle tracce di elettroni che sono prodotti nel gas dai protoni ionizzanti: essa è constante essendo il detector a gas relativamente sottile.

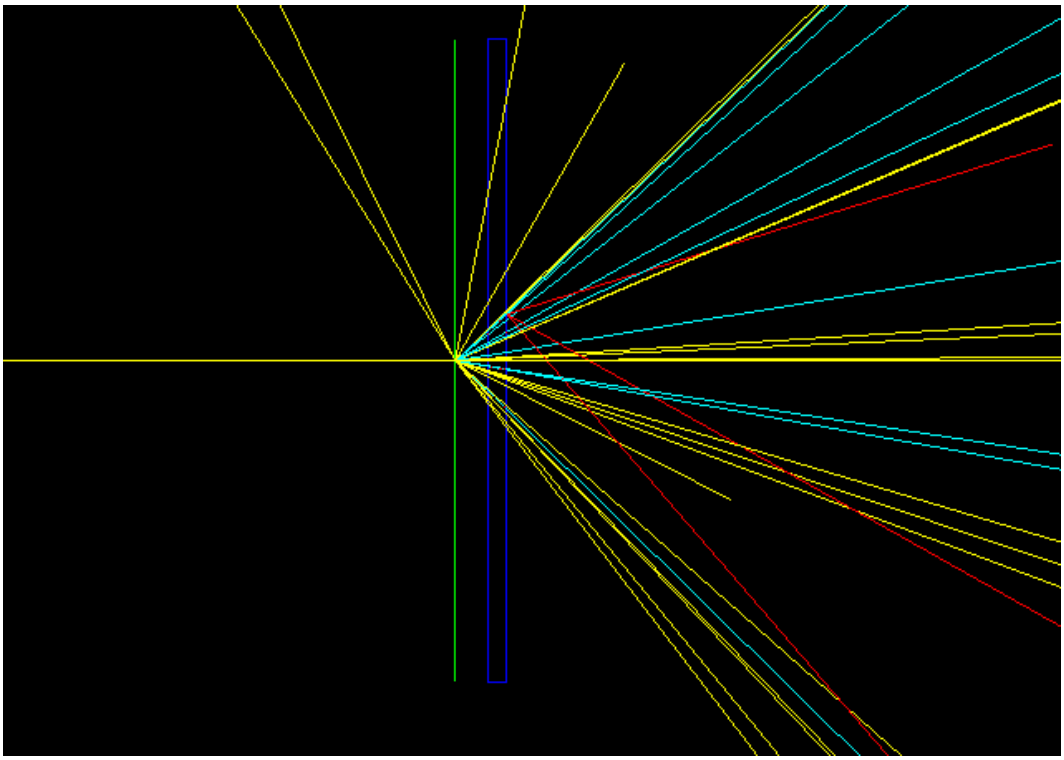

Figura 6.1: Tracce secondarie: in azzurro i protoni, in giallo i neutroni deviati dallo scattering, in rosso gli elettroni.

6.3.2 Confronto materiali

Si sono eseguiti diversi run di $1 \cdot 10^6$ neutroni di diverse energie cambiando il materiale del catodo a oro, alluminio, rame e carbonio. Si è utilizzata la seguenza macro con i comandi implementati nel `DetectorMessenger` .

```
/gps/particle neutron

/gps/energy 2.5 MeV
/det/cathodeMaterial/setMaterial G4_Al
/run/beamOn 1000000
[...]
/gps/energy 2.5 MeV
/det/cathodeMaterial/setMaterial G4_Cu
/run/beamOn 1000000
[...]
```

In figura 6.9 a pagina 59 si è confrata l'efficienza di conversione dei neutroni con i diversi materiali. In generale l'efficienza di conversione del polietilene ha un massimo maggiore di quello degli altri materiali perchè è ricco di idrogeno.

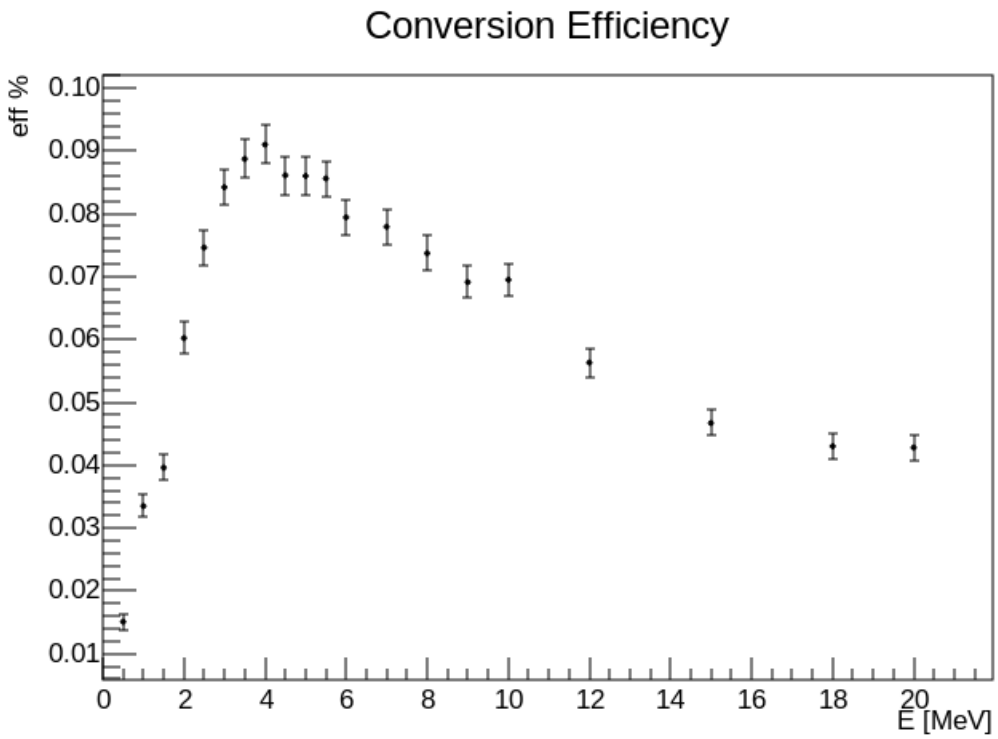

Figura 6.2: Efficienza di conversione di neutroni del polietilene

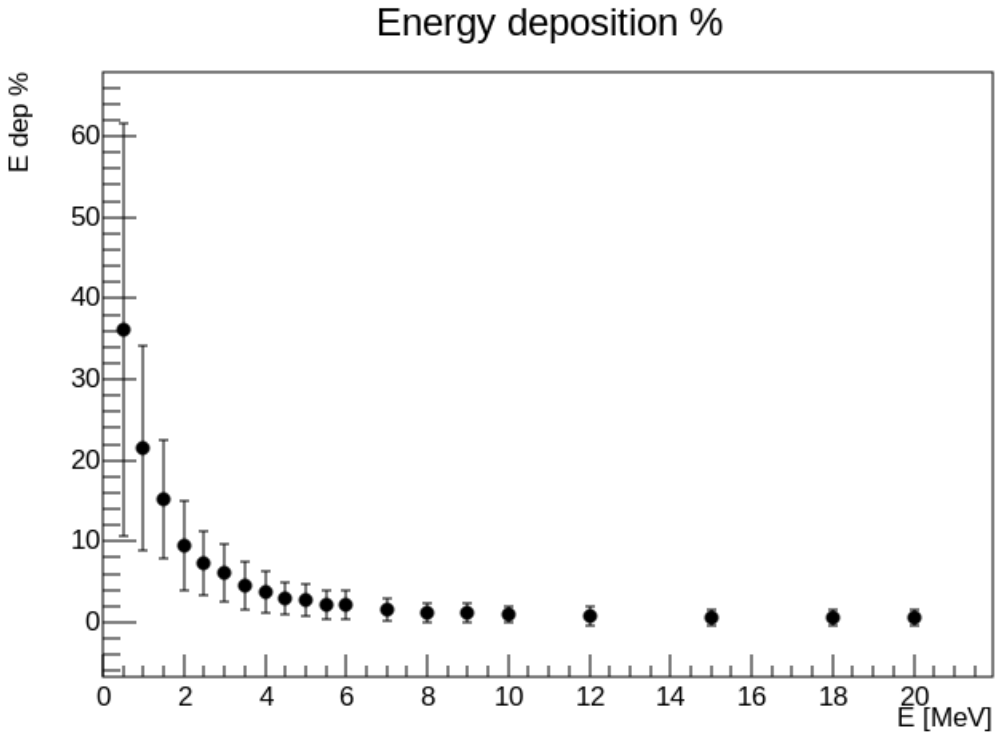

Figura 6.3: Energia depositata percentuale dalle tracce seconarie

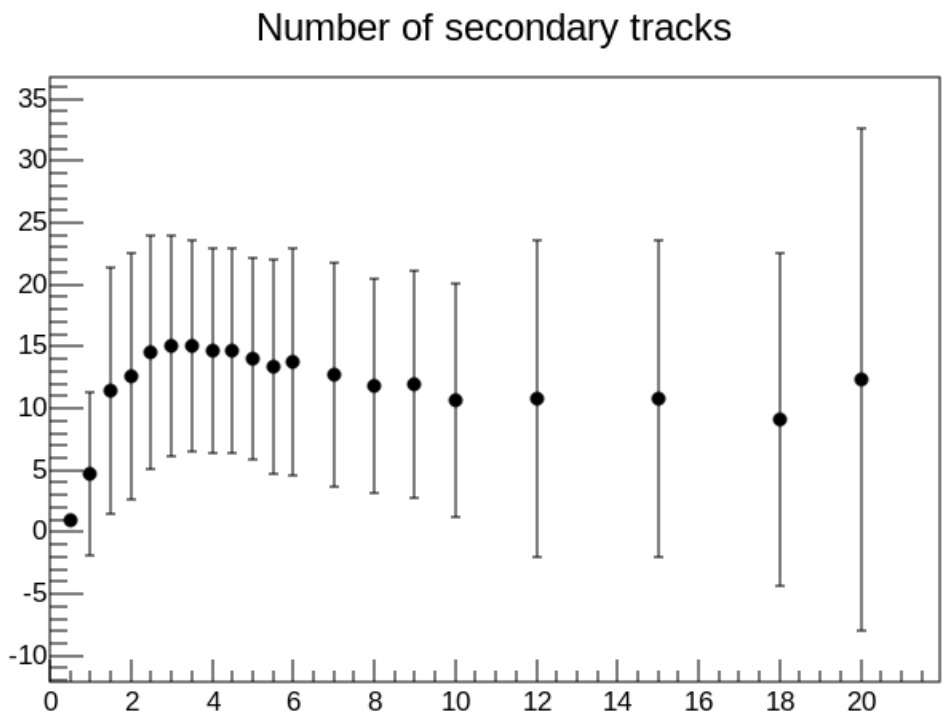

Figura 6.4: Numero medio di tracce secondario per energia dei neutroni primari

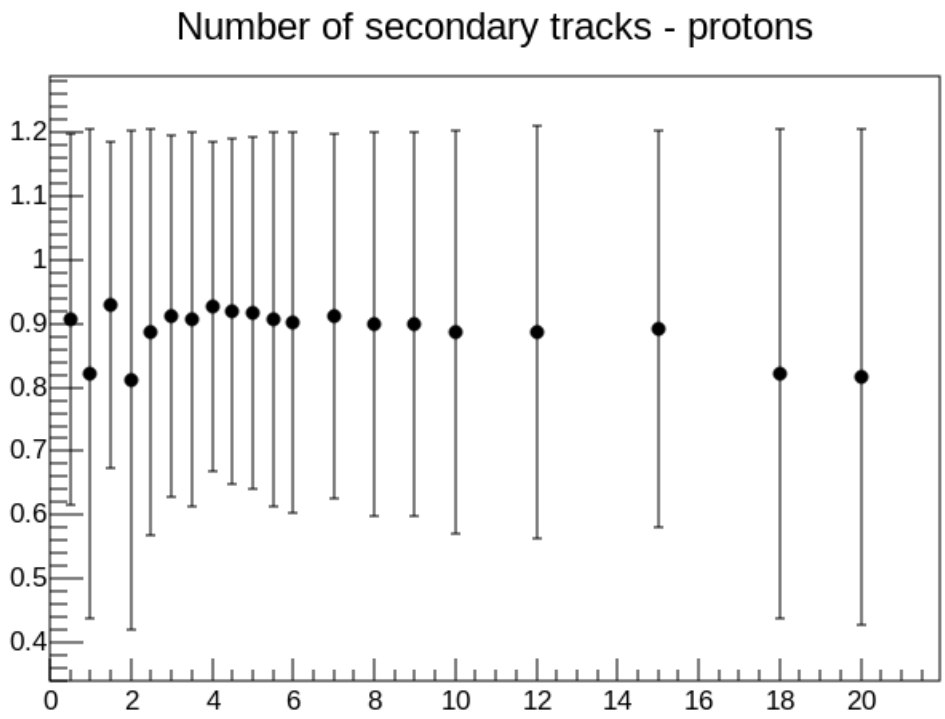

Figura 6.5: Numero medio di protoni generati in un evento in cui il neutrone ha interagito

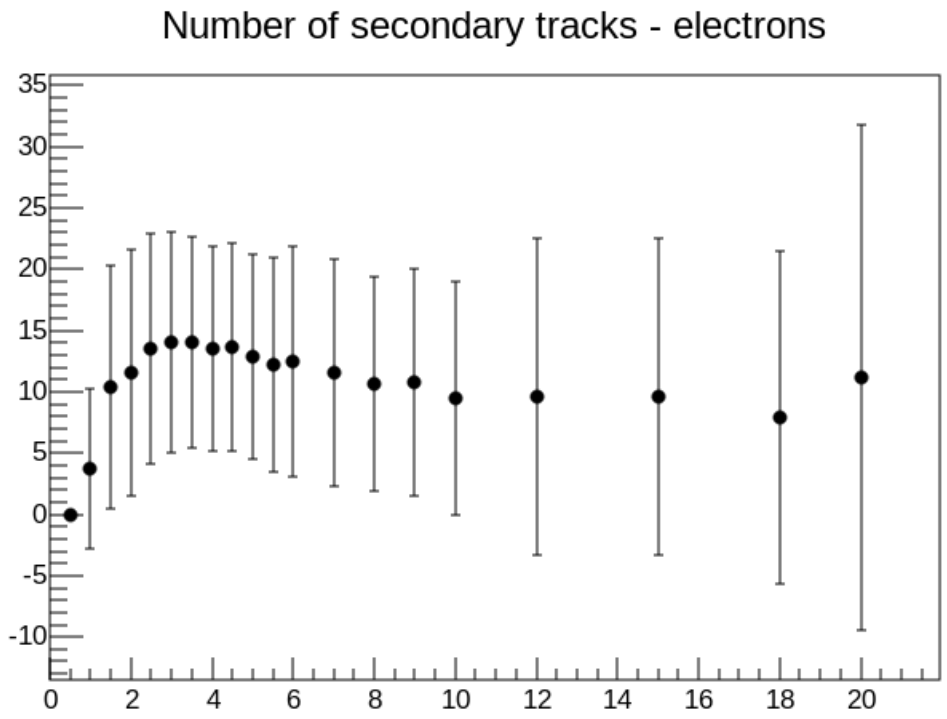

Figura 6.6: Numero medio di elettroni generati in un evento in cui il neutrone ha interagito

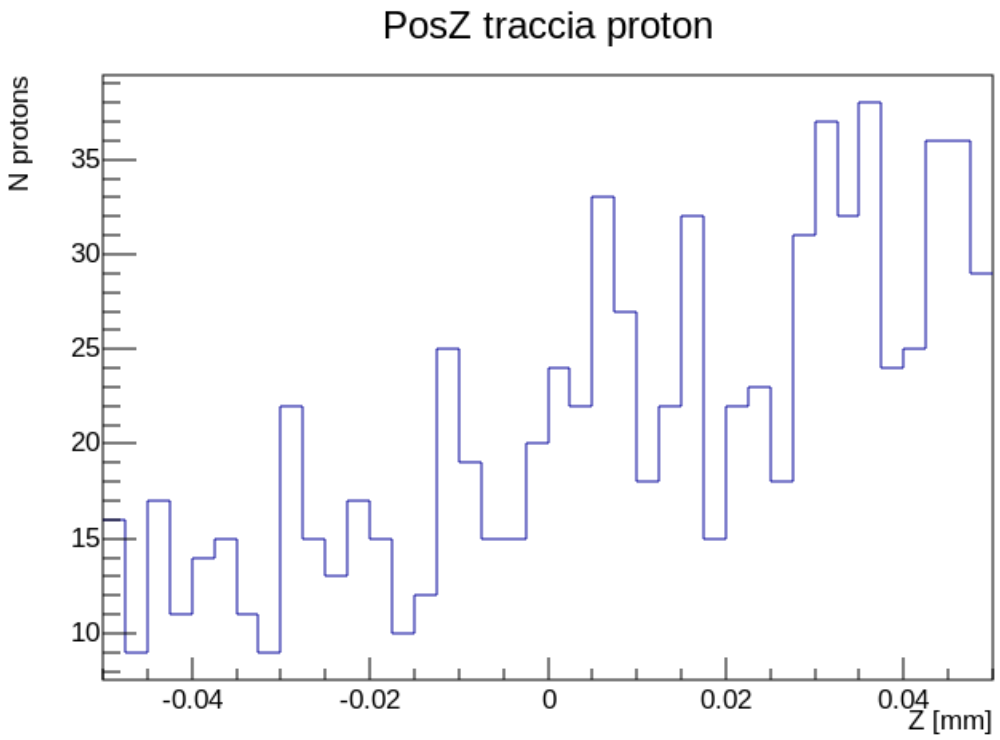

Figura 6.7: Posizione di partenza delle tracce di protoni nel catodo con fascio di neutroni da 4 MeV

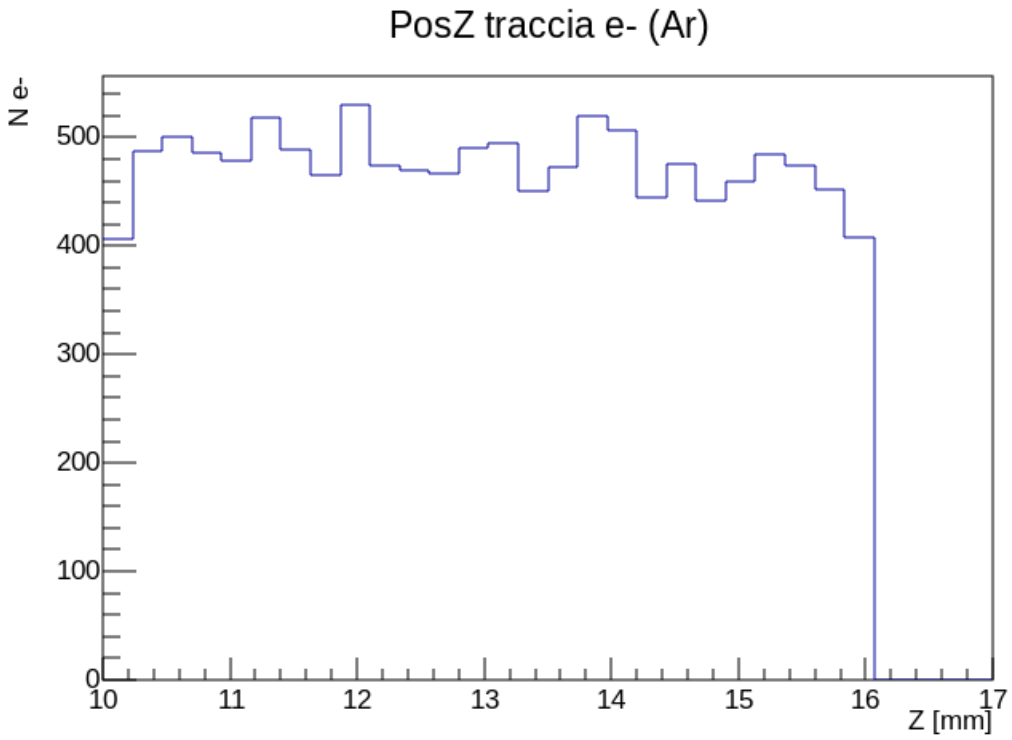

Figura 6.8: Posizione di partenza della tracce degli elettroni prodotti dalla ionizzazione del gas argon

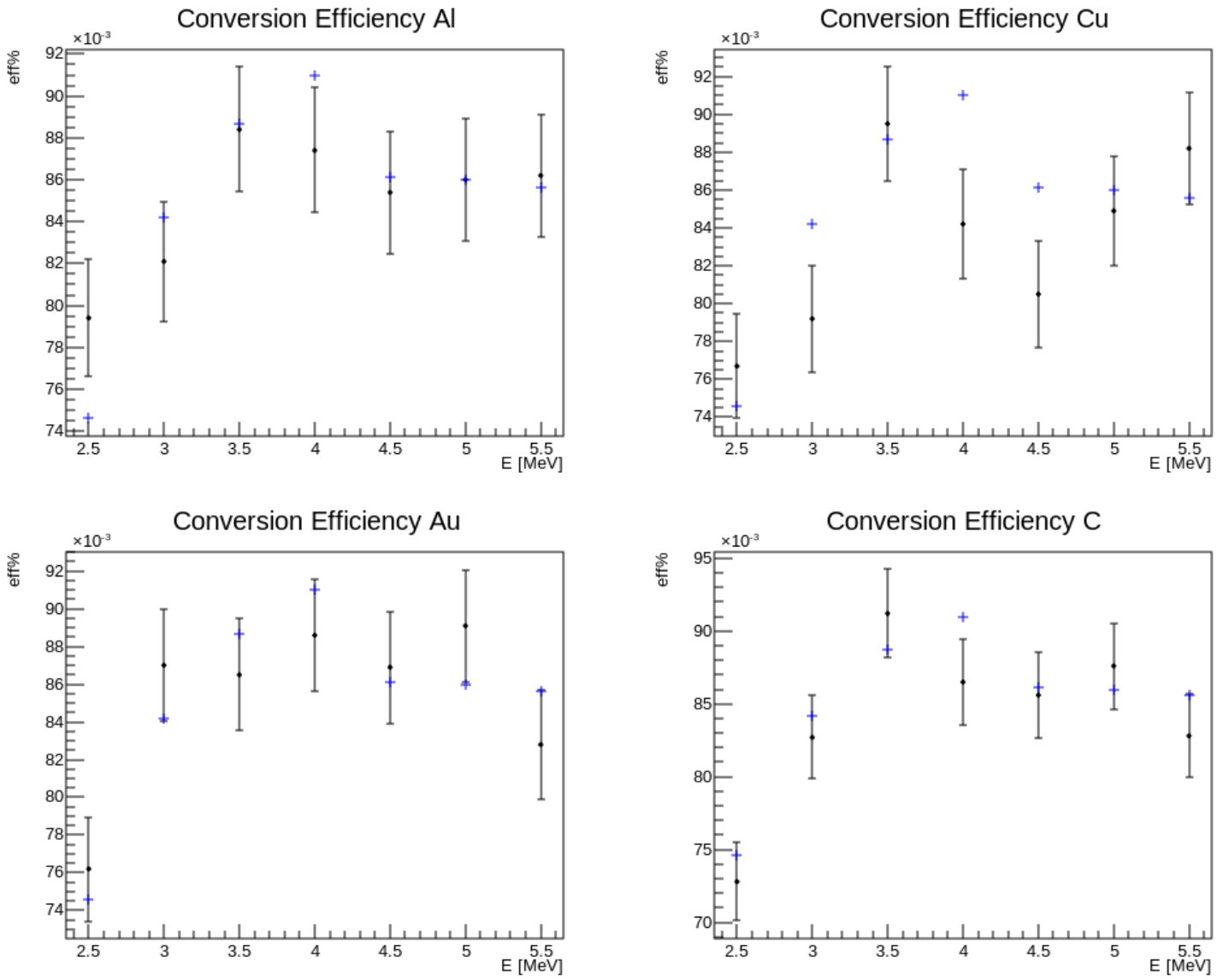

Figura 6.9: Confronto efficienza di conversione con polietilene (punti in blu)

Appendice A

Compilazione con CMake

Il processo di compilazione di un programma Geant4 è configurabile attraverso il moderno strumento `CMake` .

Questo programma permette di generare automaticamente dei `makefiles` basandosi su un set di comandi, variabili e funzioni gestibili dall'utente attraverso un file `CMakeLists.txt` . In particolare il pacchetto di Geant4 installato su una macchina Linux fornisce a `CMake` attraverso delle configurazioni di sistema, le variabili necessarie a individuare le librerie e i dati necessari per compilare una simulazione. Inoltre è possibile effettuare facilmente il linking alle librerie di ROOT a un progetto sempre attraverso Cmake.

La struttura del `CMakeLists.txt` presentata è la medesima in tutti gli esercizi.

Prima di tutto si imposta il nome del progetto e lo standard C++ richiesta (si è utilizzato C++14 sia per compilare Geant4 che ROOT).

```
# Setup the project
cmake_minimum_required(VERSION 2.6 FATAL_ERROR)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
set(CMAKE_CXX_STANDARD 14)
project(taskN)
```

Si impostano alcune variabili utilizzate come flag all'interno della simulazione attraverso la funzione di cmake `option()` e si utilizza la funzione `find_package()` per individuare l'installazione di Geant4 all'interno del sistema operativo e i relativi file di configurazione per cmake. Ad esempio si impostano le opzioni per attivare il sistema grafico.

```
option(WITH_GEANT4_UIVIS "Build example with Geant4 UI and
    Vis drivers" ON)

if(WITH_GEANT4_UIVIS)
  find_package(Geant4 REQUIRED ui_all vis_all)
else()
  find_package(Geant4 REQUIRED)
```

```
endif()

set(G4NEUTRONHP_USE_ONLY_PHOTONEVAPORATION ON)
```

Si utilizza `find_package()` anche per trovare ROOT e caricarne le impostazioni.

```
option(G4ANALYSIS_USE_ROOT "use ROOT" ON)
find_package(ROOT REQUIRED)
```

A questo punto è necessario indicare a cmake i file header da utilizzare: sia quelli di Geant4 e ROOT sia quelli del progetto. E' possibile utilizzare la variabile `${Geant4_USE_FILE}` perchè è stata impostata da `find_package`.

```
include(${Geant4_USE_FILE})

include_directories(${PROJECT_SOURCE_DIR}/include
          ${Geant4_INCLUDE_DIR}
          ${ROOT_INCLUDE_DIRS})
```

Si indicano poi a cmake i file sorgente del progetto utilizzando in comando `GLOB` che ricerca ricorsivamente file `.cc`. Si crea l'eseguibile linkando le librerie di Geant4 e ROOT:

```
file(GLOB sources ${PROJECT_SOURCE_DIR}/src/*.cc)
file(GLOB headers ${PROJECT_SOURCE_DIR}/include/*.hh)

add_executable(exetask4b task4b.cc ${sources} ${headers})

target_link_libraries(exetask4b ${Geant4_LIBRARIES} ${
    ROOT_LIBRARIES})
```

Infine si copiano le macro e si installa il programma nella cartella `bin` di riferimento.

```
set(EXAMPLETASK4B_SCRIPTS
vis.mac visQt.mac
)

foreach(_script ${EXAMPLETASK4B_SCRIPTS})
configure_file(
  ${PROJECT_SOURCE_DIR}/${_script}
  ${PROJECT_BINARY_DIR}/${_script}
  COPYONLY
)
endforeach()
```

```
12
13 # For internal Geant4 use - but has no effect if you build
       this
14 # example standalone
15 #
16 add_custom_target(task4b DEPENDS exetask4b)
17 add_definitions(-DG4ANALYSIS_USE_ROOT)
18
19 # Install the executable to 'bin' directory under
       CMAKE_INSTALL_PREFIX
20
21 install(TARGETS exetask4b DESTINATION bin)
```

Per compilare ed eseguire il programma è sufficiente eseguire le seguenti operazioni.

```
1 # Creare una cartella per il build
2 mkdir build
3 # PROJECT_DIR e' la directory che contiene il CMakeLists.txt
4 cmake PROJET_DIR
5 # compilare il programma
6 make -j 3
7 # eseguirlo
8 ./taskN
```