UNIVERSITÀ DEGLI STUDI DI CATANIA


Facoltà di Ingegneria

Corso di Laurea Magistrale Ingegneria Informatica

# ASPETTI DI SIMULAZIONE IN GEANT4 PER IL MONITORAGGIO E IL CONTROLLO DEI FUSTI RADIOATTIVI

**Tesi di Laurea**

Maria Campione
**O55000090**

Relatore:
Chiar.mo Prof. Michele Malgeri

Correlatore:
Dott. Paolo Finocchiaro

ANNO ACCADEMICO 2011 / 2012

# Indice

*Questo lavoro lo dedico alle persone*
*più importanti della mia vita senza*
*le quali non ce l'avrei mai fatta.*

# INTRODUZIONE

Il presente lavoro di tesi analizza un tema di particolare importanza dal punto di vista sociale ed economico data la sua pericolosità: il monitoraggio delle scorie radioattive nel medio-breve termine. Svolto presso i Laboratori Nazionali del Sud dell'Istituto Nazionale di Fisica Nucleare, si pone come obiettivo la realizzazione di un modello di simulazione per il monitoraggio e il controllo delle fuoriuscite nucleari dai fusti di contenimento, lo studio qualitativo e quantitativo del problema del monitoraggio e la rivelazione della radiazione gamma proveniente dai rifiuti radioattivi. In particolare, il modello di simulazione realizzato dovrà essere in grado di effettuare il conteggio delle radiazioni gamma prodotte da una sorgente di emissione, sia puntiforme che estesa, e incidenti su una geometria di fibre scintillanti costruita in polistirene e posizionata intorno alla sorgente stessa.

L'applicazione è stata realizzata avvalendosi del toolkit Geant4. In realtà esistono altri sistemi di simulazione, ma Geant, a quanto risulta, è ciò che di meglio c'è attualmente sul mercato dei packages di simulazione, visto che garantisce la simulazione di processi fisici a basse energie[29]. Offre la possibilità di simulare e seguire, con una tecnica strettamente Monte Carlo[4], le particelle prodotte nell'interazione del fascio primario fino ad energie molto basse e permette di ricostruire con precisioni maggiori rispetto a quanto consentano altri sistemi di simulazione quali FLUKA, EGS, CALOR, etc,[6] che risultano molto simili a Geant ma più adeguati per le alte energie. L'approccio del simulatore è modulare ed aperto[7] e ciò rende il metodo applicabile qualunque sia la composizione dei materiali trattati ed integrabile con le informazioni ottenute dai metodi tipicamente impiegati. Il simulatore offre un ambiente di sviluppo per un'applicazione scritta in linguaggio *C++,* che, quindi, sfrutta le caratteristiche computazionali OO del simulatore. Per tale motivo essa è costituita da classi che definiscono gli oggetti software in grado di interagire gli uni con gli altri attraverso lo scambio di messaggi.

Una volta realizzata l'applicazione, dopo una accurata analisi del problema dal punto di vista fisico, state eseguite diverse simulazioni variando di volta in volta alcuni parametri dell'applicazione stessa quali la topologia e la posizione della sorgente, il numero di particelle emesse, etc... che hanno permesso di trarre una valutazione della risposta del rivelatore, sulla base dei dati ottenuti consistenti nei conteggi delle particelle intercettate.
Le simulazioni sono state condotte distinguendo due tipi di distribuzione dei raggi gamma:

- una prima serie di simulazioni è stata eseguita impostando una distribuzione isotropica in tutto lo spazio ottenuta con una *sorgente puntiforme*, con lo scopo di valutare il numero di particelle incidenti per ricavare l'efficienza geometrica della fibra
- una seconda simulazione è stata ottenuta realizzando una distribuzione, sempre di tipo isotropica in tutto lo spazio, ma, questa volta, ottenuta con una *sorgente estesa*.

L'applicazione è stata sviluppata ed eseguita su un processore Intel Core Duo 2 Ghz, SO Linux Ubuntu 12.04 LTS utilizzando la versione del toolkit di Geant 4.9.p02, anche se attualmente è disponibile una versione aggiornata.
In definitiva quello di cui ci siamo occupati prevede: l'analisi del fenomeno fisico, le tecniche utilizzate per lo sviluppo dell'applicazione.

La tesi è organizzata nel modo seguente:

*Capitolo 1* : breve panoramica sul progetto di monitoraggio delle scorie radioattive attualmente in fase di sviluppo presso l'INFN;

*Capitolo 2 :* introduzione alla simulazione, al metodo Monte Carlo e alle sue applicazioni nell'ambito della simulazione al calcolatore e un breve cenno su alcuni simulatori che fanno uso di questi metodi;

*Capitolo 3* : descrizione dell'architettura e del design del toolkit Geant4;

*Capitolo 4* : generalità dei rivelatori utilizzati e dettagli sulla fibra scintillante;

*Capitolo 5*: sviluppo dell'applicazione: descrizione del codice delle classi di maggiore interesse che sono state implementate e alcune aggiunte riguardanti l'interfaccia, che sono state fatte per agevolare la configurazione dell'ambiente di simulazione a runtime;

*Capitolo 6*: analisi dei risultati della simulazione.

# 1 LO SMALTIMENTO DEI RIFIUTI NUCLEARI

Lo smaltimento dei rifiuti radioattivi risultanti dalla produzione nucleare di energia (e dallo smantellamento di armi nucleari) è un problema non ancora risolto, specialmente in termini di accettabilità ambientale e sociale. Quando i rifiuti condizionati vengono depositati in un sistema di smaltimento definitivo, il loro isolamento dalla biosfera deve essere assicurato per tutto il periodo in cui dura la loro pericolosità. Tale isolamento viene realizzato tramite barriere di contenimento poste in serie, la cui funzione è di impedire la diffusione degli isotopi radioattivi verso l'esterno del deposito. La prima di queste barriere è costituita dallo stesso manufatto di condizionamento, delle barriere addizionali dovranno essere fornite dal deposito stesso, e saranno di tipo artificiale o naturale, o una combinazione delle due. La sicurezza del deposito sia nel breve che nel lungo periodo si basa quindi sull'affidabilità di queste barriere addizionali, la cui natura dipende dalla severità del contenimento richiesto e da quanto a lungo dovrà essere garantito. Le considerazioni generali necessarie per la classificazione dei rifiuti (scorie) nucleari, sono:

- per quanto tempo i rifiuti resteranno ad un livello pericoloso
- qual è la concentrazione del materiale radioattivo nei rifiuti
- se i rifiuti generano calore

La persistenza di radioattività determina per quanto tempo i rifiuti devono essere gestiti. La concentrazione e la generazione di calore indicano come devono essere maneggiati. Queste considerazioni forniscono anche informazioni sui metodi idonei di smaltimento.
La classificazione [30] varia da Paese a Paese[1], ma generalmente le categorie accettate internazionalmente sono :

- rifiuti a bassissima radioattività o non radioattivi
- rifiuti a bassa radioattività

[1] *Legge 24 Dicembre 2003, n. 368*

- rifiuti a radioattività intermedia
- rifiuti ad alta radioattività

I rifiuti a bassissima radioattività o non radioattivi includono quantità trascurabili di radioattività e possono essere trattati come i rifiuti domestici.

I rifiuti a bassa radioattività includono la maggior parte dei rifiuti derivanti dal ciclo di combustibile. Comprendono carta, stracci, strumenti, vestiario, filtri e altro, che contengono piccole quantità di radioattività essenzialmente a breve vita. Non richiedono schermatura nelle fasi di maneggio e trasporto e di riduzione del volume prima dello smaltimento. Rappresentano il 90% del volume totale, ma contengono solo l'1% della radioattività complessiva.

I rifiuti a radioattività intermedia comprendono maggiori quantità di radioattività e, di norma, richiedono una schermatura. Lo schermo può essere una barriera di piombo o di acqua per proteggere dalle radiazioni penetranti come i raggi gamma. I rifiuti a radioattività intermedia comprendono essenzialmente resine, fanghi chimici, rivestimenti metallici del combustibile. Possono esser inglobati in calcestruzzo o bitume per lo smaltimento.

I rifiuti ad alta radioattività includono i prodotti di fissione e gli elementi transuranici prodotti nel reattore che sono altamente radioattivi e generano calore. Questi rifiuti rappresentano oltre il 95% della radioattività totale anche se la quantità di materiale prodotto è modesta, circa 25-30 tonnellate di combustibile spento o tre metri cubi per anno di rifiuti vetrificati per un grande reattore.

Nel caso di rifiuti a bassa attività, che costituiscono circa il 95% dell’intera produzione, l'isolamento deve essere garantito al massimo per qualche secolo (trecento anni è il tempo che determina un abbattimento dei livelli di radiazione di circa mille volte dei radionuclidi a vita più lunga come il Cs-137 o lo Sr-90). Questo è un periodo durante il quale è possibile assicurare la conservazione e la durata di barriere artificiali adeguatamente progettate e messe in opera. Infatti, lo smaltimento dei rifiuti a bassa attività è già

praticato industrialmente mediante la collocazione dei manufatti in strutture ingegneristiche di vario tipo, nella maggior parte dei casi realizzate in superficie, (trincee, silos, tumuli) costruite in calcestruzzo e con criteri che ne assicurano la conservazione in tutte le condizioni prevedibili. Le unità di deposito, una volta riempite con i rifiuti condizionati e sigillate con opportuni accorgimenti, diventano nei sistemi più avanzati dei blocchi di calcestruzzo di grande stabilità meccanica e chimica. In tal modo tra il rifiuto radioattivo e l'ambiente esterno sono interposte almeno tre barriere di protezione altamente affidabili, la cui conservazione per qualche secolo non pone problemi di sorta. La sicurezza radiologica, e cioè l'efficienza delle barriere e dell'isolamento, è continuamente controllata da sistemi e reti di monitoraggio ambientali, estesi al deposito ed alle aree circostanti, e attivi per tutto il periodo di controllo istituzionale, al termine del quale il sito viene rilasciato senza restrizioni. Oltre a ciò, i siti per la localizzazione di questi depositi vengono scelti sulla base di indagini geografiche che tengono conto di parametri generali e locali di idoneità, con l'obiettivo di conferire al sito stesso una difesa supplementare e di non favorire comunque la dispersione dei radionuclidi nell'ambiente anche dopo la fine del periodo di controllo istituzionale.

Esistono casi in cui il sito ha caratteristiche tali da non richiedere barriere artificiali di contenimento particolarmente severe, com'è il caso dei siti desertici, caratterizzati da una quasi completa assenza di precipitazioni e di falde significative. Ci sono quindi diverse strategie che permettono di trovare, o meglio, di cercare le migliori soluzioni per tenere sotto controllo le problematiche dei rifiuti nucleari[1].

Di sicuro uno dei temi di maggiore interesse di smantellamento e stoccaggio dei rifiuti radioattivi è il suo *monitoraggio nel breve, medio e a lungo termine.* Per quanto riguarda lo smaltimento nel *lungo periodo* dei rifiuti è previsto mediante l'uso di nuovi reattori che soddisfano il requisito "*pulito*", invece per quanto riguarda il monitoraggio a *breve* e *medio* termine i rifiuti sono attualmente imballati in fusti, solitamente di cemento, e mantenuti all'interno del sito di produzione o trasportati in depositi di stoccaggio dedicati.

L'obiettivo fondamentale del monitoraggio è quindi determinare e ricercare rapidamente eventuali perdite dovute alla carenza strutturale o ad incidenti[2].

## 1.1 DMNR: DATA MESH for NUCLEARE REPOSITORIES

*SISTEMA PER IL MONITORAGGIO REAL-TIME DELLE SCORIE RADIOTTIVE*

DMNR è un progetto che mira alla realizzazione di un prototipo capace di implementare il monitoraggio *on-line di breve-medio termine* dei rifiuti radioattivi, attualmente in sviluppo presso INFN-LNS Catania[2]. Tale sistema risulta avere le seguenti caratteristiche:

- distribuito
- scalabile
- robusto
- affidabile
- basato su componenti a basso costo.

Da un punto di vista tecnico il monitoraggio delle scorie radioattive era già possibile anni fa, anche se con alcune limitazioni. Infatti i sistemi di monitoraggio presenti già in diverse località all'interno dei siti di stoccaggio, come ad esempio i contatori Geiger[31] pur essendo dei buoni dispositivi di controllo hanno presentato un limite piuttosto importante dovuto al loro costo, ecco il perchè della nascita di questo progetto nel quadro di una collaborazione tra INFN e Ansaldo Nucleare S.p.a.. Gli obiettivi di tale progetto sono monitoraggio in tempo reale: attività, stabilità meccanica, corrosione, temperatura, etc. , disponibilità dei dati in tempo reale alle autorità di controllo, quali vigili del fuoco, governo locale e nazionale, etc. e gestione dei contenitori per mezzo di strumenti avanzati e con procedure atte a ridurre i rischi ai lavoratori locali, alla popolazione circostante, all'ambiente.

[2] E.N.E.A. Agenzia Nazionale per le nuove tecnologie, l'energia e lo sviluppo economico sostenibile

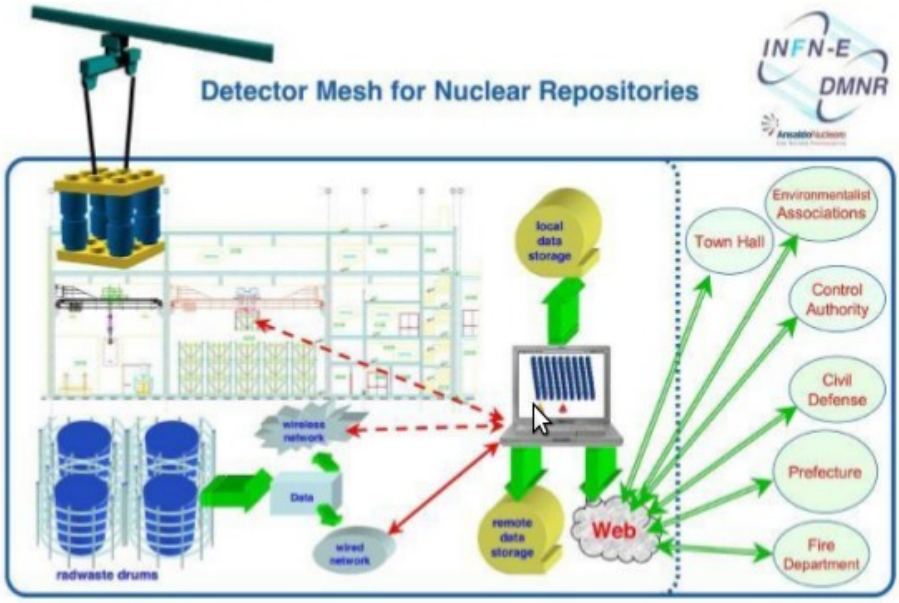



*Fig. 1: Prototipo Detector Mesh for Nuclear Repositories*

Il prototipo è caratterizzato da una fitta maglia di rivelatori da installare sui bidoni contenenti i rifiuti nucleari. Il flusso di dati provenienti dai sensori verso una consolle è gestito attraverso un'elettronica di front-end e un sistema di conteggio basati su FPGA[3] (Field Programmable Gate Array), tale sistema si occupa della trasmissione dei dati ridondanti, di una interfaccia utente grafica e un sistema di archiviazione dati. In questo sistema è anche prevista la possibilità di effettuare un'ispezione remota ad hoc attraverso un braccio robotico, dotato sia di una telecamera per l'ispezione visiva, che di rivelatori di radiazione ad alta risoluzione. I risultati dei test effettuati con sorgenti radioattive hanno mostrato prestazioni molto promettenti in termini di sensibilità, tali da incoraggiare l'implementazione di un piccolo sistema dimostrativo per il monitoraggio reale di alcuni fusti di rifiuti radioattivi recentemente testato nel deposito della ex centrale del Garigliano in provincia di Caserta.

[3] In elettronica digitale il Field Programmable Gate Array è un circuito integrato programmabile il cui comportamento viene definito via software dall'utente finale

## 1.2 DECADIMENTO ATOMICO E RADIAZIONI

Prima di parlare più dettagliatamente dello sviluppo della nostra applicazione è necessaria una breve descrizione del decadimento atomico e delle sue caratteristiche. Quando un nucleo decade, si trasforma in un nucleo di un elemento diverso. Questo è esattamente ciò che volevano realizzare gli alchimisti dei secoli passati e che in realtà avviene spontaneamente in natura!
Esistono due tipi fondamentali di decadimento:

- il *decadimento alfa* in cui l'atomo si spezza perdendo 2 protoni e 2 neutroni, cioè un nucleo di $^4$He. Questo nucleo di elio sparato via ad alta velocità viene definito radiazione alfa. Inoltre spesso il nucleo principale viene a trovarsi in uno stato eccitato, dopo essersi spezzato, e ritorna ad uno stato di minore energia emettendo onde luminose di brevissima lunghezza d'onda, dette radiazione gamma.
- il *decadimento beta*, in cui uno dei neutroni del nucleo si trasforma in protone, con emissione di elettroni molto energetici (radiazione beta), di antineutrini e di radiazione gamma. Dei neutrini non dobbiamo preoccuparci, visto che interagiscono pochissimo con la materia, mentre gli altri tre tipi di radiazione sono molto pericolose per la salute, anche se il loro comportamento è differenziato.

I *raggi alfa*[4] sono molto energetici ed in grado di provocare seri danni alle strutture molecolari incontrate. Tuttavia, interagendo molto con la materia, non sono in grado di attraversare grandi spessori e possono essere schermati abbastanza facilmente. Posti a contatto con strutture viventi sono però molto pericolosi, specie se il contatto è prolungato nel tempo. I raggi alfa hanno carica positiva e sono quindi deviati dai campi magnetici.
I *raggi beta*[5], essendo costituiti da elettroni, sono più penetranti e più difficili da schermare ed anche loro in grado di provocare seri danni alle strutture biologiche. Avendo carica negativa, sono anche loro deviati dai campi magnetici.

[4]Le **particelle alfa**, **raggi alfa** o **elioni** sono una forma di radiazione corpuscolare altamente ionizzante e con un basso potere di penetrazione dovuto all'elevata sezione d'urto
[5]La radiazione beta è una forma di radiazione ionizzante emessa da alcuni tipi di nuclei radioattivi come il cobalto-60. Questa radiazione assume la forma di particelle beta (β), che sono elettroni o positroni ad alta energia.

I *raggi gamma*[6] sono radiazione elettromagnetica di brevissima lunghezza d'onda e sono molto penetranti, nonché in grado di danneggiare le molecole organiche. Essendo neutri, non sono deviati dai campi magnetici.

Le tipologie di radiazioni emesse da un fusto di rifiuti radioattivi sono: alfa, beta, gamma e neutroni dovuti a fissione nucleare. La fissione nucleare[7] è il processo in cui un nucleo pesante si spezza in due frammenti (tipicamente radioattivi), in genere liberando al tempo stesso un paio di neutroni. Le particelle alfa, beta e i prodotti radioattivi della fissione nucleare a causa della loro massa non riescono a fuoriuscire dal fusto bloccandosi all'interno, grazie all'impiego del materiale isolante; i neutroni, prodotti anch'essi da fissione nucleare, sono altamente penetranti ma vengono prodotti in piccole dosi. Per quanto riguarda i gamma, essi non sono altro che fotoni ad alta energia estremamente penetranti; per tale motivo fuoriescono dal fusto e costituiscono l'effettiva radiazione in uscita dai fusti di scorie radioattive.


*Fig. 2: Schema delle radiazioni e del loro livello di penetrazione*

[6]In fisica nucleare i raggi gamma (spesso indicati con la lettera greca minuscola gamma, γ) sono una forma di radiazione elettromagnetica prodotta dal cosiddetto decadimento gamma o da processi nucleari o subatomici consistenti dunque nell'emissione di fotoni ad alta energia.
[7]In fisica nucleare la **fissione nucleare** è una reazione nucleare, comunemente utilizzata nei reattori nucleari, in cui il nucleo di un elemento passante decade in frammenti di minori dimensioni, ovvero in nuclei di atomi a numero atomico inferiore, con emissione di una grande quantità di energia e radioattività.

## 1.3 MONITORAGGIO DELLE RADIAZIONI PROPOSTO DAL DMNR

La soluzione proposta dal DMNR riguarda un dispositivo a basso costo per il conteggio dei raggi gamma che si comporta come un contatore Geiger-Müller scintillante[1]. Il sistema di monitoraggio, progettato per essere distribuito, robusto, affidabile e basato su componenti poco dispendiosi, propone un rivelatore scintillante, equipaggiato con dei fotosensori SiPM[7] (Silicon Photomultiplier) in grado di rilevare piccoli segnali di luce dovuti a pochi fotoni. Ciascuno di questi sensori viene replicato al fine di costituire una griglia attorno ad ogni fusto di rifiuti: è possibile posizionare facilmente diversi sensori di radiazione attorno ad ogni fusto in varie posizioni, registrando continuamente l'attività misurata al fine di conoscere istantaneamente il tasso di radiazione e la sua storia.

I risultati dei test effettuati con sorgenti radioattive hanno mostrato prestazioni molto promettenti in termini di sensibilità, tali da incoraggiare l'implementazione di un piccolo sistema dimostrativo per il monitoraggio reale di alcuni fusti di rifiuti radioattivi recentemente testato con successo.

Per confermare le aspettative sulla sensibilità della fibra scintillante adoperata in questo tipo di dispositivi sono stati condotti alcuni test tramite misure di precisione e simulazioni: il sensore, costituito dalla fibra e da due SiPM alle sue estremità, è stato sottoposto ad una sorgente con bassa attività come il $^{60}$CO (cobalto- 60)[8] o il $^{137}$CS (cesio-137)[9].

La sorgente è stata posta a contatto con la fibra in tre diverse posizioni, come mostrato in Fig. 3 nel deposito della ex centrale del Garigliano in provincia di Caserta.

[7]Il Silicon Photomultiplier è un dispositivo sensibile al passaggio dei singoli fotoni costituito da un fotodiodo a valanga costruito su un substrato di silicio.

[8]Il $^{60}$CO è un isotopo radioattivo del metallo cobalto che decade per decadimento beta nell'isotopo stabile nichel-60. Il nucleo di nichel attivato emette due raggi gamma con energie di 1,17 e 1,33 MeV (Megaelettronvolt).

[9]Il $^{137}$CS è un isotopo radioattivo del metallo alcalino cesio che si forma principalmente come un sottoprodotto della fissione nucleare dell'uranio. Tale isotopo va incontro a decadimento beta ed emette raggi gamma di 662 keV.

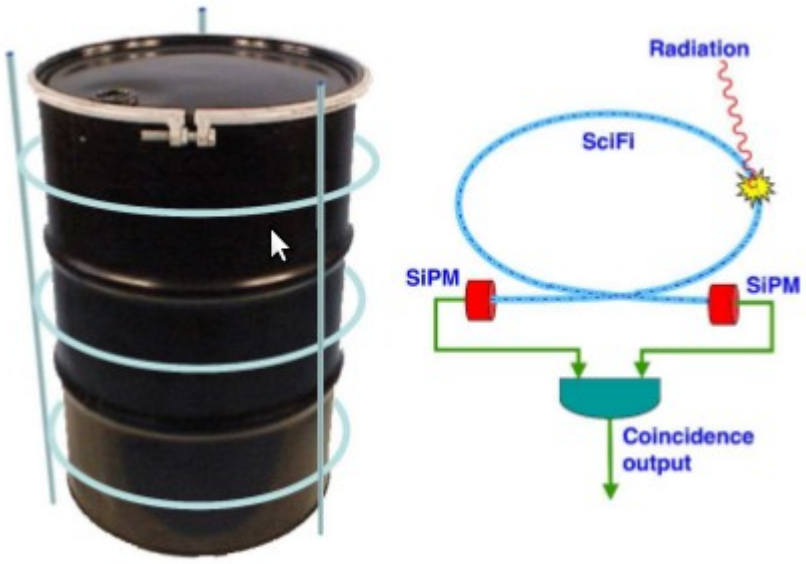



*Fig. 3: Fibra attorno al fusto (a sinistra), e uno schema del principio di funzionamento del sensore (a destra).*

L'attendibilità del dispositivo di monitoraggio è presentato nei seguenti grafici, che mostrano come effettivamente la fibra sia sensibile alla radiazione gamma, infatti, in corrispondenza della sorgente è visibilmente maggiore il numero di gamma conteggiati[2].

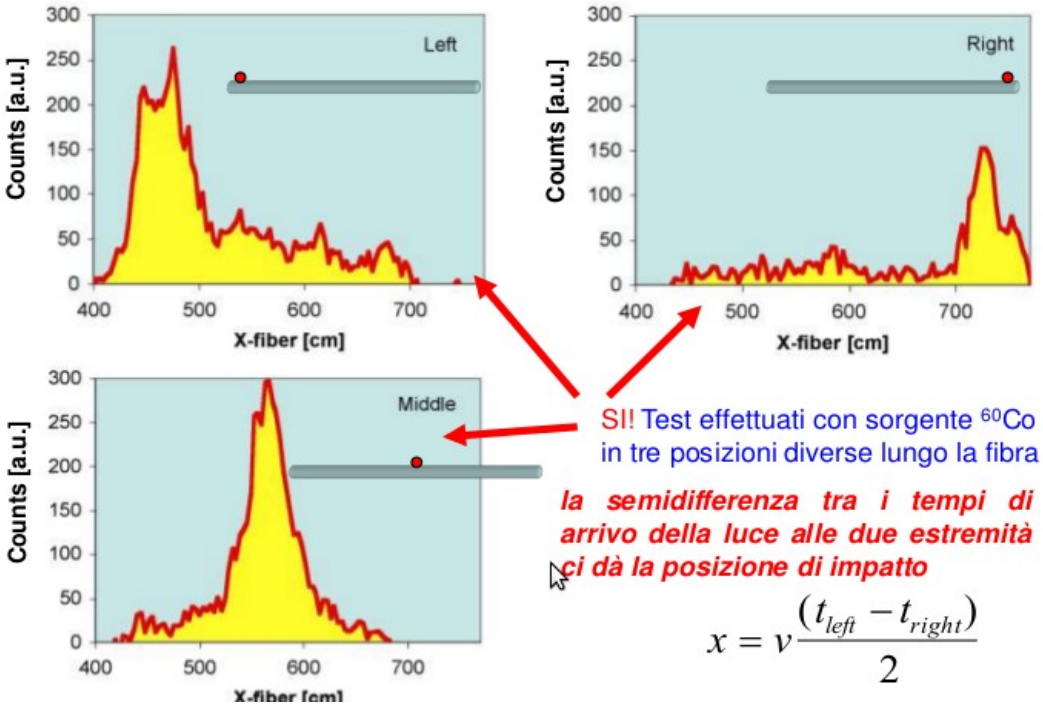



*Fig. 4: Sensibilità della fibra e del SIMP*

La sensibilità della fibra e del SIMP sono state verificate mediante opportune simulazioni realizzate da prototipi in sviluppo all'INFN. In particolare essa è stata valutata sottoponendo la fibra alla radiazione della sorgente, posta di volta in volta a diverse distanze mediante spostamenti micrometrici. I risultati ottenuti sono stati soddisfacenti in quanto confrontabili con quelli in possesso del laboratorio, ottenuti sia da test concreti che da implementazioni del codice fatte per mezzo della precedente versione del toolkit GEANT, ma questo verrà trattato in dettaglio nel prossimo capitolo.

Sulla base delle informazioni acquisite riguardo il monitoraggio dei rifiuti radioattivi a breve e medio termine adottato all'INFN, si svilupperà un simulatore in C++ basato su Geant4 per il monitoraggio dei fusti di contenimento dei rifiuti nucleari[10].

Vediamo in dettaglio le proprietà geometriche e fisiche degli oggetti caratterizzanti il simulatore:

- il *fusto di contenimento* delle scorie radioattive è costruito in *cemento* ed ha la forma di un *cilindro equilatero*, la cui altezza e il diametro sono pari ad 1 m;
- il *sistema di monitoraggio* è realizzato disponendo quattro fibre scintillanti intorno al fusto. Ciascuna fibra[11] è costituita di due parti, una parte esterna detta *cladding* costruita in *plexiglass* e una parte interna detta *core* costruita in *polistirene*, quest'ultima è la parte sensibile del rivelatore. Le fibre hanno tutte un diametro di 1mm e lunghezza di 1.2 m.
- la *sorgente di radiazioni gamma* rappresenta le possibili fuoriuscite dal bidone di contenimento. È realizzata da un emettitore a bassa attività di $^{137}$CS mediante due tecniche di emissione: sorgente puntiforme ad emissione isotropica in tutto lo spazio e sorgente estesa.

[10] Rif. Pag 9

[11] Per maggiori informazioni vedi capitolo 4

## 2 LA SIMULAZIONE

La simulazione permette di valutare un processo attraverso un altro, dove per processo si intende una sequenza temporale degli stati di un sistema[3]. Secondo questa definizione la simulazione trova applicazione in diversi ambiti, da quello scientifico a quello economico, etc. e in generale in tutti quei campi in cui un certo fenomeno può essere descritto attraverso un modello. L'importanza della simulazione risiede anche nel fatto di costituire un potente strumento di analisi e verifica del sistema oggetto di osservazione; quest'ultimo può essere analizzato con un elevato livello di dettaglio trascurando tutti quegli effetti inutili o poco significativi ai fini dello studio che si intende realizzare, semplificando così la complessità del sistema, consentendo di concentrarsi esclusivamente sugli effetti di principale interesse.

Tipicamente le potenzialità della simulazione risiedono nella capacità di calcolo dei computer che costituiscono il mezzo principale attraverso cui tale strumento trova applicazione. In generale il ruolo della simulazione è la descrizione dell'evoluzione temporale di un sistema reale attraverso un modello computazionale. Essa viene applicata nell'ambito della ricerca per eseguire esperimenti numerici che permettono di estrapolare dati e informazioni difficilmente ottenibili attraverso esperimenti reali a causa dell'eccessiva complessità analitica del sistema o per impossibilità di eseguire materialmente l'esperimento a causa di limiti teorici, pragmatici o etici. In altre parole, "*la simulazione al computer costituisce un nuovo approccio metodologico in ambito scientifico che si pone a livello intermedio tra la teoria e il metodo empirico di condurre esperimenti e osservazioni: la sperimentazione numerica*"[12]. In ambito scientifico questo strumento costituisce un supporto essenziale per gli esperimenti consentendo un significativo risparmio di risorse in termini di tempo e costi.

[12] Cit[3]

È ad esempio possibile conoscere gli effetti conseguenti a differenti configurazioni dei parametri di un sistema simulato in tempi sicuramente più brevi rispetto al caso reale: i risultati dettagliati delle simulazioni, svolte valutando tutte le possibili impostazioni dell'esperimento, possono essere disponibili prima ancora che l'esperimento sia realmente eseguito. Esistono diverse tecniche di simulazione e un vantaggio indiscutibile della simulazione è quello di dare una risposta a molti problemi di calcolo e scientifico che si dimostrano intrattabili se affrontati con altri approcci. Si può affermare che un processo di simulazione consiste nel:

- costruire un modello che sia in grado di funzionare nel tempo in modo da imitare da vicino il sistema in esame
- generare dal modello numerosi campioni di casi possibili e studiarne il comportamento al trascorrere del tempo
- analizzare i risultati evidenziando le decisioni alternative.

Si possono distinguere almeno tre categorie fondamentali di modelli di simulazione:

- i modelli icastici che sono delle ricostruzioni in scala dei sistemi che si vogliono studiare;
- i modelli analogici che sono di tipo fisico e i cui parametri sono legati matematicamente ai parametri del sistema in oggetto di studio;
- i modelli logico-matematici (detti anche modelli simbolici o astratti) che consentono di creare una storia completa del sistema, riproducendolo mediante un insieme di proposizioni logiche e di relazioni di natura matematica e statistica.

In questa sede ci occuperemo esclusivamente di un particolare insieme di modelli logico–matematici, analizzando quella procedura che in letteratura è chiamata *tecnica di simulazione Monte Carlo*.

## 2.1 LA SIMULAZIONE MONTE CARLO

E' un potente metodo di simulazione che si fonda sulla generazione di numeri casuali per rappresentare il comportamento di un fenomeno caratterizzandolo attraverso la sua statistica[4]. Nacque negli anni della seconda guerra mondiale, quando un gruppo di scienziati, ingegneri e tecnici lavoravano alla nascita di uno dei primi calcolatori elettronici, chiamato ENIAC (Electronic Numerical Integrator And Computer). Questa macchina, costruita da Prosper Eckert e John Mauchly, il quale tra l'altro lavorava ai contatori Geiger[31] a cui si è accennato nel primo capitolo, suscitò l'interesse di molti scienziati i quali compresero subito che la potenza di calcolo poteva essere sfruttata per far eseguire alla macchina calcoli complessi come la risoluzione di equazioni differenziali. Il primo a sfruttare le potenzialità del calcolatore ENIAC per lo studio delle reazioni termonucleari fu John Von Neumann, che a quel tempo lavorava al progetto Manhattan[4] per la costruzione della bomba atomica. Agli sviluppi dell'ENIAC, che fu completato nella primavera del 1946, parteciparono diverse personalità scientifiche tra cui Enrico Fermi, Nicholas Metropolis, Edward Teller e in seguito Stanislaw Ulam. Quest'ultimo, grazie alle sue conoscenze matematiche, comprese che la potenza di calcolo dei calcolatori poteva essere sfruttata per resuscitare tecniche statistiche cadute in disuso a causa della lunghezza e della noia nei calcoli: fu così che, avendo proposto la sua idea a Von Neumann, si innescò la scintilla che condusse i due allo sviluppo del metodo Monte Carlo. Già, quindici anni, prima Enrico Fermi aveva applicato tecniche di campionamento statistico per studiare le proprietà del neutrone e fu di fatto il primo ad utilizzare questo metodo.

Con il termine metodo Monte Carlo si fa riferimento ad una classe di algoritmi computazionali per la risoluzione di sistemi complessi difficilmente risolvibili con metodi analitici, laddove entra in gioco la necessità di studiare il sistema assumendo decisioni basate sulla distribuzione di probabilità di un certo fenomeno, al fine di generare l'evoluzione di un possibile scenario, statisticamente valido, del sistema in esame e trarre conclusioni da esso. Le decisioni da assumere sugli eventi che intervengono durante l'evoluzione del sistema oggetto di studio sono formulate sulla base di una distribuzione uniforme di numeri pseudo-casuali generati dal calcolatore attraverso un algoritmo di generazione. La distribuzione di numeri pseudo-random ottenuta dovrà poi essere trasformata in una distribuzione non uniforme opportuna che descriva la proprietà di interesse del fenomeno che si intende studiare. Il campionamento casuale trova anche applicazione nella risoluzione di integrali complessi[13].

[13] Rif. [4][5]

## 2.2 Monte Carlo

Per sua natura il metodo Monte Carlo è particolarmente adatto allo studio di fenomeni stocastici[5] e quindi trova impiego nell'ambito della simulazione al fine di riprodurre lo stato di un sistema mediante procedure numeriche. Il metodo può essere applicato a qualsiasi fenomeno generando eventi secondo la distribuzione di probabilità. Tramite il calcolatore è possibile simulare per migliaia di volte il processo aleatorio, ricostruendo un numero a piacere di realizzazioni possibili del fenomeno con l'ausilio delle tecniche Monte Carlo e raccogliendo così rapidamente una serie di dati che, trattati con metodi statistici, forniscono stime tanto più attendibili quanto più è grande il numero delle realizzazioni valutate. Questo tipo di simulazione, denominata simulazione stocastica o simulazione Monte Carlo, si attua riproducendo il meccanismo preso in esame, sostituendo alla valutazione analitica l'osservazione empirica: i risultati prodotti saranno sensati se il valore medio delle misure condotte sulle realizzazioni del processo simulato convergerà al valore reale.

Affinché un sistema possa essere studiato mediante l'uso del metodo Monte Carlo esso deve poter essere descritto da *pdf* (funzioni densità di probabilità)[14].
Note le *pdf* che descrivono il sistema si procede al loro campionamento casuale, effettuato generando numeri random compresi nell'intervallo[0,1]. Per avere un risultato significativo si deve svolgere un gran numero di simulazioni e da esse ricavare un valore medio della grandezza da misurare associando l'errore statistico. L'elemento principale per una simulazione Monte Carlo è il generatore di numeri random. In realtà si tratta di numeri pseudo-random: se l'algoritmo di generazione inizia sempre dallo stesso punto di partenza, la sequenza di numeri ottenuta sarà identica, tuttavia questo

[14] Nella teoria della probabilità la pdf (probability density function) di una variabile casuale continua è una funzione che descrive la probabilità che la variabile casuale assuma un determinato valore. La probabilità che il valore assunto dalla variabile casuale si trovi in un determinato intervallo è data dall'integrale della pdf della variabile in questo intervallo. La funzione densità di probabilità è non negativa e il suo integrale esteso in tutto lo spazio è uguale a uno.

costituisce un vantaggio in quanto rende l'esperimento “riproducibile”, caratteristica senza la quale non sarebbe possibile ripetere esattamente la stessa simulazione e quindi individuare gli errori connessi alla scrittura dell'algoritmo[6].

## 2.3 STATO DELL'ARTE

Esistono diversi simulatori che utilizzano questa tipologia di metodi numerici, in particolare nell'ambito della fisica delle particelle, come:

- *FLUKA* (FLUktuierende KAskade): codice Monte Carlo integrato per la simulazione del trasporto e dell'interazione di particelle e nuclei; trova molte applicazioni nel campo della fisica delle particelle elementari, nel calcolo di radioprotezione, fisica dei raggi cosmici, dosimetria, fisica medica e soprattutto in radioterapia; è sviluppato in linguaggio FORTRAN ed è disponibile sotto forma di libreria di oggetti pre-compilati per alcune piattaforme di calcolo. Il codice sorgente può essere reso disponibile alle condizioni specificate nella licenza di FLUKA. Il software è distribuito e mantenuto dall'INFN e dal CERN che ne detengono il copyright. A volte viene impropriamente utilizzato il nome FLUKA per riferirsi a precedenti versioni della sola parte relativa al generatore di interazioni adroniche che sono state interfacciate ad altri codici[6]. In particolare per Geant3. In questo caso si dovrebbe usare come riferimento, per esempio, il nome Geant-FLUKA. Infatti la versione del generatore adronico di FLUKA implementato in Geant3 è del 1993 e non è mai stato ulteriormente sviluppato[7].

- *MCNP* (Monte Carlo N-Particle Transport Code): è un package per la simulazione di processi nucleari; è sviluppato dal Los Alamos National Laboratory ed è distribuito negli Stati Uniti dal Radiation Safety Information Computational Center e in ambito internazionale dal Nuclear Energy Agency; è usato principalmente per la simulazione di

processi nucleari, come la fissione, ma ha anche la capacità di simulare l'interazione di particelle che coinvolge neutroni, fotoni ed elettroni; le aree di applicazione includono radiation protection e dosimetria, fisica medica, nuclear criticality safety, progetto di reattori di fusione, etc.

- *EGS* (Electron Gamma Shower): è un package general purpose per la simulazione Monte Carlo del trasporto di coppie di elettroni e neutroni; questo toolkit per la simulazione è scritto in linguaggio Mortran3; è stato sviluppato da A. James Cook e L. J. Shustek allo Stanford Linear Accelerator Center (SLAC) e tutte le sue caratteristiche sono state incluse in Geant3.

- *GEANT* (*Geometry and Tracking*) software di simulazione progettato per descrivere il passaggio di particelle elementari attraverso la materia utilizzando metodi Monte Carlo. Il nome è un acronimo che sta per “geometria e tracciamento”. Originariamente sviluppato al CERN per esperimenti di fisica delle alte energie, oggi viene usato in molti altri campi. La prima versione di GEANT risale al 1974; versioni a partire dalla 3.21 sono state scritte in FORTRAN e mantenute come parte di CERNLIB. Dal 2000 circa, l'ultima release FORTRAN riceve solo correzioni occasionali. GEANT3, tuttavia, è ancora in uso da alcuni esperimenti; è disponibile sotto la licenza GNU (General Public License), con l'eccezione di un codice di interazione adronica, contributo della collaborazione FLUKA. L'ultima versione del software è Geant4; si tratta di una completa riscrittura in C++ con un moderno design object-oriented. Geant4 è stato sviluppato dalla collaborazione RD44 nel 1994-1998 ed è stato mantenuto e migliorato dalla collaborazione Geant4. Superata una fase in cui non ha avuto una licenza software ben definita, a partire dalla versione 8.1 (rilasciata il 30 Giugno 2006) Geant4 è disponibile con la “Geant4 Software License”. Attualmente nel sito

web (http://geant4.cern.ch/) sono disponibili le versioni 4.9.4p04 rilasciata il 13 aprile 2012 e la 4.9.5 rilasciata il 23 maggio 2012.

Per lo sviluppo della nostra applicazione è stata utilizzata la versione 4.9.4p01. Come già detto, supporta la simulazione di numerose implementazione caratterizzate da svariate esigenze, come le applicazioni spaziali e dei raggi cosmici, i calcoli nucleare, ioni pesanti e radiazioni, e le applicazioni mediche. Rappresenta, inoltre, il programma leader per la simulazione della risposta dei rivelatori sensibili[7]. Le sue caratteristiche modulari permettono la gestione e la produzione di software distribuito in una collaborazione internazionale, con la partecipazione agli esperimenti di numerosi istituti e laboratori[14]. Fornisce il diritto al *Production Service* e *User Support* per tutti i fisici INFN, per tutti gli esperimenti e progetti di interesse per l'INFN, offre la possibilità di contribuire alle decisioni tecniche e scientifiche della Collaborazione Geant4, è particolarmente adatto alle applicazioni di Fisica alle basse energie, agli esperimenti di astroparticelle/applicazioni spaziali e applicazioni bio-mediche[8]. Di Geant4 parleremo ampiamente nel capitolo che segue.

[14] Istituti che utilizzano Geant: SLAC, CERN, CEA, KEK, INFN, Manchester University STFC, etc...

## 3. GEANT

Geant4 [14] è l'acronimo di *GEometry ANd Tracking, geometria e tracciamento* è un toolkit per la "simulazione del passaggio di particelle attraverso la materia"[13], fornisce un insieme completo di strumenti per tutte le aree di simulazione tipiche di queste applicazioni: geometria e risposta di rivelatori, tracciamento, gestione di run, evento e traccia, visualizzazione ed interfaccia utente. Un'ampia varietà di processi fisici tratta le interazioni di particelle con la materia su un vasto intervallo di energie. Le principali potenzialità di Geant4 sono il trasporto di particelle attraverso l'apparato sperimentale, la simulazione della risposta dei rivelatori posti lungo la traiettoria e la rappresentazione grafica dell'apparato sperimentale e delle traiettorie delle particelle[32].

### 3.1 DESIGN E ARCHITETTURA

Il toolkit di simulazione Geant4, proposto ed approvato dal DRDC (Detector R esearch and Development Committee) del Cern (Laboratorio Europeo per la Fisica delle Particelle Elementari) alla fine del 1994 è stato sviluppato nel periodo 1994-1998 da RD44[24], una collaborazione internazionale costituita da un centinaio di fisici, informatici ed ingegneri provenienti da più di quaranta istituzioni europee, americane, russe, canadesi e giapponesi.

La prima versione di produzione è stata pubblicata nel dicembre 1998; con questa RD44 ha concluso il suo mandato, ed è stata creata una nuova Collaborazione internazionale Geant4[25], regolata da un *Memorandum of Understanding* (MoU), per la gestione della fase di produzione. Il codice e la documentazione di Geant4 sono pubblicamente disponibili su Web [26].

Attualmente sono membri della Collaborazione Geant4: CERN, KEK, SLAC, TRIUMF, IN2P3, LEBE- DEV (Accademia delle Scienze Russa), ESA, ATLAS, CMS, LHCB, BABAR e il Gruppo Italian Geant4 Developers [27].

[13] Rif. [14], pag14

Geant4 è interamente scritto in *C++* ed utilizza la tecnologia Object-Oriented per una maggiore chiarezza delle implementazioni fisiche. Nell'analisi Object-Oriented si utilizzano le categorie *classe* dette anche *domini* per creare unità logiche. La decomposizione in domini l'analisi e il design orientati all'oggetto hanno condotto ad una chiara struttura gerarchica dei sottodomini, caratterizzata da un flusso unidirezionale di dipendenze. Una categoria contiene classi che hanno tra loro strette relazioni; classi appartenenti a differenti domini hanno legami molto deboli e solo un numero limitato di queste hanno relazioni con le altre categorie. I domini utilizzati in Geant4 sono rappresentati nella figura sottostante. Ogni box della figura rappresenta un dominio e le relazioni tra i domini sono rappresentati da una linea continua. Il cerchio al termine di ogni linea indica che il dominio che ha il cerchio utilizza altri domini. I principali domini di Geant4 sono:

- *Run ed Event*, legati alle generazione degli eventi. Il loro scopo principale è quello di fornire le particelle che saranno poi seguite con il Tracking Management.
- *Tracking* e *Track*. Sono i domini legati alla propagazione di una particella analizzando i fattori che limitano lo step e applicando i fattori fisici più rilevanti.
- *Geometry* e *Magnetic Field.* Queste due categorie gestiscono la definizione della geometria di un rivelatore.
- *Particle Definition e Matter* : gestiscono la definizione dei materiali che costituiscono il rivelatore e le particelle utilizzate nella simulazione.
- *Physics*. Questa categoria controlla tutti i processi fisici che intervengono nelle interazioni delle particelle con la materia. L'interfaccia dei processi fisici permette implementazioni multiple di vari modelli che possono essere selezionati per range, tipo di particella, materiali utilizzati.
- *Hits* e *Digitization*. Sono i domini che controllano la creazione di Hits ed il loro uso in fase di digitizzazione.
- *Visualization*. Questa categoria è predisposta alla visualizzazione della geometria, delle traiettorie seguite dalle particelle, dei punti di interazione; essa interagisce con le librerie gra...che di base come OpenGL, DAWN, Postscript e VRML.

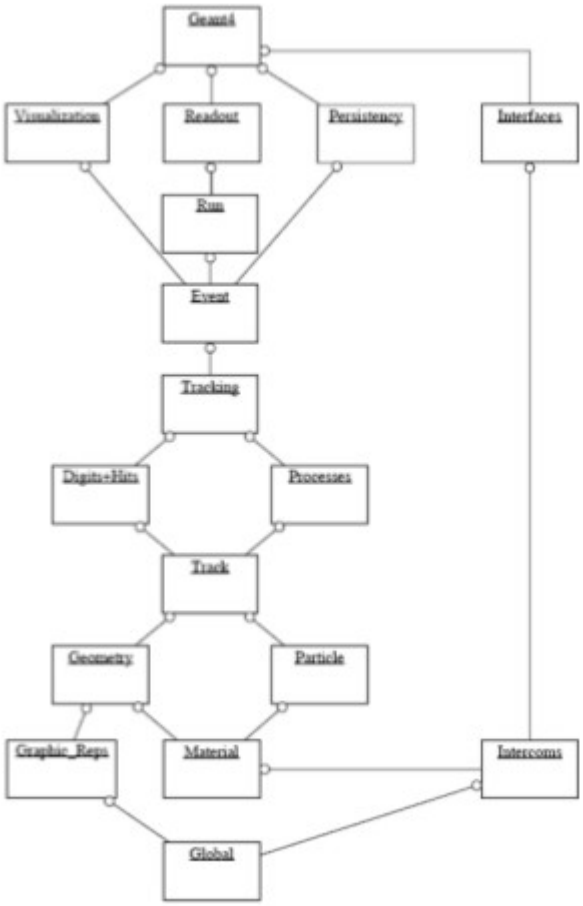



*Fig. 5: Diagramma Rappresentazione a blocchi dei domini di GEANT4 e loro relazioni.*

### 3.1.1 USER INTERFACE

La categoria *intercoms* fornisce un mezzo per interagire con Geant4 attraverso l'interfaccia utente e un modo per far comunicare tra loro i vari moduli. Questa categoria permette all'utente di interagire con il programma di simulazione attraverso l'uso di comandi già implementati o da implementare, divisi in directory in base alle funzionalità loro assegnate. L'utente, come si vedrà più avanti, può definire nuovi comandi, creando una propria classe "*Messenger*". Alcune tra le directory di comandi messe a disposizione dal toolkit sono:

- */run/*: contiene i comandi legati allo svolgersi del run. Tra questi i principali sono initialize, per inizializzare il kernel di Geant4; beamOn che fa partire il run; verbose per indicare le informazioni da visualizzare sul run.
- */tracking/*: contiene i comandi relativi alla traiettoria della particella e agli step. Il comando abort, interrompe il corrente processo G4Track e resume lo riattiva; storeTrajectory per memorizzare e visualizzare o no le traiettorie; verbose che indica il livello di informazioni sulle tracce delle particelle che devono essere visualizzate.
- */particle/*: In questa directory troviamo i comandi relativi alle particelle incidenti: select permette di selezionare una particella; list, stampa la lista delle particelle disponibili in Geant; find per trovare la particella tra quelle disponibili.
- */vis/*: contiene i comandi di visualizzazione, divisi per directory in base alla funzione loro assegnata. Si possono così definire lo zoom, l'angolazione, i colori, ecc.
- */gun/*: gestisce i comandi del fascio incidente, il tipo di particella, la direzione, il punto di partenza, l'energia cinetica, ecc. Il comando list stampa la lista delle particelle disponibili; number definisce il numero di particelle da generare; random lancia in modo casuale la particella incidente.
- */gps/* : gestisce i comandi per la generazione di sorgenti estese, il tipo di particella, la direzione, il punto di partenza, ecc. Il comando type permette di definire il tipo di sorgente; shape permette di definire la forma della sorgente.

### 3.2 DEFINIZIONE DELLA GEOMETRIA

La rappresentazione geometrica degli elementi in Geant4 si concentra sulla definizione di modelli solidi, sulla loro posizione spaziale nonché sulle loro relazioni logiche, definiamo, perciò, gli aspetti che riguardano la creazione dei volumi componenti un rivelatore in Geant4[18].

Come prima cosa deve essere creato il volume "*world*" che conterrà tutti gli altri volumi definiti per la costruzione del rivelatore, in maniera tale da costituire una struttura gerarchica in cui ogni volume può essere collocato all'interno di un altro, indicato come "*volume madre*", che dovrà essere più

grande, perchè i volumi non devono sovrapporsi, e in riferimento al quale dovrà essere fatto il posizionamento del volume contenuto, indicato come "*volume figlio*". Il volume *world* costituisce il volume di riferimento per le applicazioni Geant4 e non è contenuto all'interno di nessun altro volume, è posizionato nell'origine del sistema di coordinate globale e rappresenta il sistema di riferimento per tutti gli altri volumi creati in seguito. Ogni volume è il risultato della composizione di tre oggetti:

- *solido*: l'istanza di un "solido" *(G4Box, G4Tubs, etc.)* rappresenta un oggetto geometrico con una data forma e dimensioni (per esempio il volume madre è costruito come un cubo con un lato di 10 centimetri, il fusto come un cilindro di raggio 1 m e altezza 1 m, ma questi li esamineremo più tardi);
- *logico*: un "volume lo logico: un "*volume logico*" (istanza della classe G4LogicalVolume) specifica le caratteristiche fisiche, che comprendono le proprietà geometriche del volume, contenute nell'istanza del solido creato in precedenza, che dovrà essere passato come parametro, e il materiale di cui il volume deve essere composto
- *fisico*: un "volume fisico" (G4PVPlacement) rappresenta il posizionamento spaziale di un volume logico. È possibile posizionare una singola copia del volume oppure copie ripetute: nel caso del posizionamento di una singola copia si definiscono una matrice di rotazione e un vettore di traslazione per il volume da posizionare, che rappresentano rispettivamente la rotazione e la traslazione del volume relativamente al sistema di riferimento del "volume madre" in cui il volume sarà contenuto [18].

### 3.3 DEFINIZIONE DEI MATERIALI CHE COSTITUISCONO I VOLUMI

In Geant4 sono implementate tre importanti classi per la definizione dei materiali (elementi, materiali semplici, misure) e precisamente:

- la classe *G4Element* descrive le proprietà degli atomi: numero atomico, numero di nucleoni, massa atomica etc.;
- la classe *G4Material* descrive, invece, le proprietà macroscopiche della materia: densità, stato, temperatura, pressione etc.;

- la classe *G4Isotope* che definisce i diversi isotopi in base al loro numero ed alla massa di una mole.

Tutte queste classi, se le frazioni in massa dei vari elementi costituenti sono definite correttamente, calcolano e forniscono per ogni materiale una tabella delle sezioni d'urto, della perdita di energia e dei range per alcune delle più importanti particelle. La definizione degli elementi può essere fatta conoscendo il peso atomico a dell'elemento, il numero atomico Z, il simbolo chimico ed il nome dell'elemento. È possibile anche la definizione di altri materiali per i quali si conosce la composizione chimica o la frazione in massa degli elementi.

### 3.4 SETUP DI UNA APPLICAZIONE

Creare una applicazione in Geant4 vuol dire derivare ed implementare alcune classi *concrete*, partendo dalle classi astratte fornite dal kernel del toolkit. Questa operazione è possibile grazie alle proprietà della programmazione OO che permettono il *polimorfismo* ed il *dynamic-binding*. Esistono due tipi di classi: quelle obbligatorie *(mandatory user classes)* e quelle opzionali *(optional user classes)*. Le classi obbligatorie sono tre, di cui due sono definite user *initialization classes*, utilizzate per l'inizializzazione dell'applicazione, ed una *user action class*, utilizzata per l'inizializzazione dell'esecuzione.
Le classi astratte da cui derivare le proprie classi sono:

- *G4VUserDetectorConstruction:* nella classe derivata da questa classe astratta è necessario definire l'area di funzionamento della simulazione con la descrizione di tutte le geometrie, i materiali, le aree sensibili e le superfici.

- *G4VUserPhysicsList:* nella classe derivata da questa classe astratta è necessario definire tutte le particelle ed i processi fisici che interagiranno durante la simulazione.

- *G4VUserPrimaryGenerationAction:* questa è la user action class e nella classe derivata da questa è necessario specificare il modo in cui devono essere generate le particelle primarie.

L'esistenza di queste tre classi viene verificata all'inizio della simulazione dal G4RunManager al momento dell'invocazione dei metodi *initialize()* e *beamOn()*.
Le classi opzionali, invece, permettono all'utente la modifica e la personalizzazione del comportamento di default di Geant4. Le cinque principali *optional user classes* sono:

- *G4UserRunAction*: per impostare azioni da eseguire all'inizio ed alla fine di ogni run della simulazione.

- *G4UserEventAction*: per impostare azioni da eseguire all'inizio ed alla fine di ogni evento della simulazione.

- *G4UserStackingAction*: per personalizzare l'accesso allo stack in cui vengono memorizzate le informazioni di tracciamento delle particelle.

- *G4UserTrackingAction*: per impostare azione da eseguire all'inizio alla creazione ed al completamento di ogni tracciato.

- *G4UserSteppingAction:* per personalizzare le azioni da eseguire ad ogni step del processo di simulazione.

# 4 I RIVELATORI

I rivelatori di particelle utilizzati negli esperimenti di fisica nucleare si sono nel tempo accresciuti in dimensioni, sensibilità e complessità, con un conseguente lievitare dei costi complessivi. È uno strumento usato per rivelare, tracciare e identificare particelle, come quelle prodotte per esempio da un decadimento nucleare, dalla radiazione cosmica, o interazioni in un acceleratore di particelle. Spesso i rivelatori sono usati come calorimetri, cioè sono in grado di misurare l'energia della radiazione persa nel rivelatore.

## 4.1 Generalità

Il principio di funzionamento di tutti i rivelatori di particelle si basa sul trasferimento di tutta o di una parte dell'energia della radiazione da rivelare alla massa del rivelatore, dove viene convertita in qualche altra forma più accessibile alle percezioni umane: la forma in cui l'energia viene convertita dipende dal tipo di rivelatore. Così, per esempio, nelle emulsioni fotografiche la ionizzazione delle particelle interagenti produce reazioni chimiche che permettono la formazione di immagini, mentre nei rivelatori a gas produce ioni che danno origine ad un segnale elettrico, negli *scintillatori* induce transizioni energetiche molecolari che risultano nell'emissione di luce. La proprietà di un rivelatore di produrre in uscita un segnale utilizzabile per una data radiazione e per un dato intervallo di energia si definisce *"Sensibilità"*. Allo stato attuale non esistono rivelatori in grado di essere sensibili a tutte le radiazioni ed a tutte le energie. In generale quindi si può dire che la sensibilità di un rivelatore, per una determinata radiazione e per un determinato intervallo di energie, dipende dalla sezione d'urto di ionizzazione della radiazione nella massa sensibile del rivelatore, dalla massa totale attiva e dal materiale inattivo che circonda il materiale attivo del rivelatore stesso.
Tutti i rivelatori moderni forniscono essenzialmente un segnale in uscita di tipo elettrico, cioè l'informazione dal rivelatore è trasferita in impulsi elettrici che poi sono processati da opportuni circuiti elettronici. Oltre a rivelare la

"*presenza*" di una radiazione (sistemi di conteggio della radiazione incidente), la maggior parte dei rivelatori è anche in grado di fornire informazioni sull'energia della radiazione. Solitamente, infatti, la ionizzazione prodotta dalla radiazione in un rivelatore è proporzionale all'energia che essa deposita nel volume sensibile. Se il rivelatore è sufficientemente grande da assorbire completamente la radiazione, allora questa ionizzazione fornisce una misura dell'energia della radiazione.

La relazione esistente tra l'energia della radiazione e l'integrale del segnale di uscita dal rivelatore dà la "*Risposta*" di quest'ultimo.

I rivelatori di radiazione possono essere suddivisi in diverse categorie a seconda degli scopi per i quali vengono impiegati.

- I *"rivelatori integrali"* servono per la misura delle proprietà globali del campo di radiazione in una zona dello spazio.
- I *"contatori"* permettono di contare il numero di corpuscoli della radiazione, senza fornire informazioni sulla loro energia e spesso anche sul tipo di radiazione.
- I *"contatori differenziali"* o *"spettrometri"* non soltanto permettono di contare il numero di corpuscoli della radiazione, ma sono in grado di misurare (con precisione più o meno grande) l'energia che essi lasciano nel materiale sensibile del rivelatore e di distinguere il tipo di radiazione. Alcuni rivelatori possono essere impiegati come "rivelatori integrali" e "contatori differenziali", altri rivelatori sono al tempo stesso "contatori" e "contatori differenziali". Le emulsioni fotografiche e le camere a ionizzazione sono gli esempi più importanti di rivelatori integrali. Il più noto dei contatori semplici è il contatore Geiger[14][9]. Appartengono alla categoria dei contatori differenziali i contatori proporzionali, gli scintillatori, i rivelatori a stato solido e tutto l'insieme di spettrometri magnetici. Si analizzano nel seguito le principali caratteristiche che contraddistinguono un rivelatore.

[14] Appartengono alla categoria dei contatori differenziali i contatori proporzionali.

## 4.2 RIVELATORE A SCINTILLAZIONE

Uno scintillatore è un materiale che emette impulsi di luce quando viene attraversato da una radiazione ionizzante (come fotoni di alta energia o particelle cariche). Per questa loro caratteristica gli scintillatori trovano impiego in fisica come rivelatori di particelle, soprattutto fotoni o particelle cariche di alta energia, come nel caso di misure di radioattività. Un rivelatore a scintillazione, o contatore a scintillazione, consiste in un materiale scintillante accoppiato con un sensore di luce elettronico, come un fotomoltiplicatore o un fotodiodo, in grado di assorbire la luce emessa dal materiale scintillante e convertirla in forma di elettroni per effetto fotoelettrico, generando così un impulso elettrico che potrà essere elaborato per condurre analisi e ottenere informazioni sulla particella che ha colpito lo scintillatore stesso[10].

In natura esistono vari tipi di materiali scintillanti che si possono distinguere in 6 categorie: *cristalli organici, liquidi organici, plastici, cristalli inorganici, gas e vetri*. Gli scintillatori si possono caratterizzare, oltre che per il tipo di materiale, per i tempi di risposta, le lunghezze d'onda emesse, l'efficienza di scintillazione (quanta energia viene convertita in luce) etc. Di seguito sono descritte le sei categorie di scintillatori sopra elencate, distinte nelle due categorie principali:

1. scintillatori organici

2. scintillatori inorganici.

### 4.2.1 Scintillatori organici

- *Cristalli organici*: Si tratta di molecole organiche dotate di anelli aromatici, in cui la radiazione carica incidente eccita modi rotazionali o vibrazionali. Sono in forma solida, appunto di cristallo, e si distinguono per la risposta veloce (tipicamente entro circa un nanosecondo). Tuttavia presentano scarsa lavorabilità poiché risulta difficile organizzarli nella forma che si ritiene più opportuna, essendo essi cristallini.

- *Liquidi organici*: Le molecole sono le stesse del tipo precedente, ma invece che essere cristallizzate, sono mantenute in soluzione. Le prestazioni dipendono dalla purezza e dalla concentrazione della soluzione.

- Scintillatori plastici: Simili ai precedenti, ma il "solvente" è solido, essendo costituito da un materiale plastico facilmente lavorabile, ad esempio polistirene. Il risultato è uno scintillatore dalle buone prestazioni, anche se leggermente più lento dei precedenti (tempi di risposta di due o tre nanosecondi).

### 4.2.2 Scintillatori inorganici

- *Cristalli inorganici*: Sono composti in genere da alogenuri alcalini, ad esempio NaI. Si distinguono per l'elevato potere d'arresto (stopping power), che li rende particolarmente adatti a rivelare radiazione penetrante e per l'alta efficienza. Sono tuttavia molto più lenti degli altri, avendo tempi di risposta dell'ordine delle centinaia di nanosecondi. I cristalli sono spesso drogati, per esempio lo ioduro di sodio con il tallio. Queste impurità sono essenziali per aumentare l'efficienza di scintillazione, ridurre l'autoassorbimento e avere la luce in uscita nella lunghezza d'onda voluta.

- *Vetri*: Si tratta di scintillatori costituiti da silicati, con aggiunta di altri materiali. Essi sono principalmente utilizzati per la rivelazione dei neutroni, ma sono sensibili anche alla radiazione $\beta$ e $\gamma$. Essi sono noti per la loro resistenza a tutti i reagenti organici e inorganici eccetto l'acido idrofluoridrico. Hanno un alto punto di fusione e sono estremamente resistenti; possono quindi essere utilizzati quando si deve lavorare con agenti chimici corrosivi o con alte temperature, malgrado la loro uscita in luce sia piuttosto bassa, circa il 25-30 % dell'antracene. La loro velocità di risposta è intermedia tra quella degli scintillatori plastici e quella dei cristalli organici, essendo di poche decine di ns. Per basse energie di neutroni è possibile accrescere la sensibilità arricchendo la componente

di Litio con 6Li. Specifiche tecniche elettroniche permettono di distinguere gli eventi dovuti ai neutroni dalla radiazione γ.

- *Gas*: sono principalmente i gas nobili: xenon, kripton, argon ed elio nonché Azoto. In questi scintillatori gli atomi vengono individualmente eccitati dalla radiazione o dalla particella incidente, ritornando al loro stato fondamentale in circa un ns; pertanto la loro risposta è estremamente rapida. Essi emettono nell'ultravioletto, che è la regione in cui la maggior parte dei fotomoltiplicatori sono inefficienti. Per poter ottenere una discreta efficienza di conversione nel fotocatodo occorre rivestire le pareti interne del recipiente contenitore, con un shifter di lunghezza d'onda (guide di luce laccate con materiali opportuni, che hanno la capacità di assorbire radiazione di una determinata lunghezza d'onda e riemetterla in un altro range di frequenza). Gli scintillatori gassosi sono generalmente usati in esperimenti con particelle cariche pesanti o frammenti di fissione. In questi esperimenti sono state utilizzate misture di molti gas sotto pressione fino a 200 atm per accrescerne l'efficienza. Negli ultimi anni gli scintillatori gassosi sono stati proposti come rivelatori in fisica spaziale.

Gli scintillatori più comuni usati per rivelare radiazione sono cristalli inorganici, materiali organici, plastici e liquidi. La maggior parte sono cristalli inorganici o plastici, il più comune è lo ioduro di sodio drogato con tallio, che ha un'alta efficienza di scintillazione[11].

### 4.3 FIBRA SCINTILLANTE

Le fibre scintillanti sono sostanzialmente dei fili di scintillatore plastico di diametro variabile fra un centinaio di micron e qualche millimetro. Il limite principale è la lunghezza delle fibre. Inevitabilmente infatti la luce che viaggia nella fibra subisce qualche fenomeno che ne diminuisce l'intensità: fibre troppo lunghe potrebbero non produrre agli estremi alcun segnale. Le fibre scintillanti comunemente usate sono costituite principalmente da polistirene (PS), una molecola complessa contenente un anello di benzene. Il core, ovvero la parte propriamente attiva e che funge da rivelatore, è contenuto in una guaina (*cladding*), che ha uno spessore totale di qualche micron ed un indice di rifrazione inferiore all'indice di rifrazione del *core*: questo sia per aumentare l'efficienza di intrappolamento, sia per proteggere la fibra. Il *polystyrene* presenta una densità di 1.032 g/cm$^3$, è costituito da due diversi elementi (carbonio ed idrogeno) presenti in uguali quantità nella molecola, la cui formula bruta è $(C_8H_8)_n$. Il plexiglass ha una densità di 1.18 g/cm$^3$, è costituito da tre elementi: carbonio, idrogeno ed ossigeno. In ogni molecola, la cui formula bruta è$(C_5O_2H_8)_n$ , sono presenti 5 atomi di carbonio, 2 di ossigeno e 8 di idrogeno. Quelle comunemente usate sono costituite principalmente da polistirene, una molecola complessa contenente un anello di benzene.

Il corpo di polistirene è contenuto in una o due guaine (cladding) questo sia per aumentare l'efficienza di intrappolamento sia per proteggere la fibra. Le particelle ionizzanti che attraversano il corpo della fibra interagiscono con la molecola di polistirene: in particolare la molecola di benzene contenuta, per la sua configurazione elettronica, si presta in maniera ottimale all'assorbimento di energia.

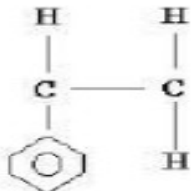



*Fig. 6: Molecola di Polystyrene, l'agglomerato esagonale rappresenta la molecola di benzene contenuta.*

La soluzione proposta dal DMNR per il sensore di radiazione consiste di fibre di plastica scintillanti dello spessore di 1 mm accoppiate a dei fotomoltiplicatori al silicio (SiPM)[15] (Silicon Photomultiplier). Le fibre di plastica sono note per essere piuttosto rigide alle radiazioni contro invece la debolezza dei SiPM ma in grado di rilevare anche piccoli segnali di luce.
Tuttavia, alcuni progressi hanno dimostrato che l'inserimento dei SiPM incapsulati in un conduttore risultano essere una buona soluzione.
Questo nuovo tipo di rivelatore è stato implementato con lo scopo principale di effettuare il conteggio di radiazioni gamma così che si comporta un contatore di Geiger-Müller[16] i cui fotosensori sono in grado di rilevare i piccoli segnali luminosi di pochi fotoni. Il Contatore Geiger-Müller[8] è l'attuale sistema di monitoraggio dei rifiuti capace dei effettuare il conteggio di particelle cariche: alfa, beta, protoni e generalmente particelle ionizzanti. Quando una particella ionizzante attraversa il dispositivo, determina una breve scarica elettrica che, opportunamente amplificata, viene segnalata dallo strumento che è quindi in grado di rilevare il numero di particelle presenti. Una soluzione del genere per il monitoraggio real-time attorno ad ogni singolo fusto avrebbe però un costo proibitivo. Invece il sistema di monitoraggio in questione propone un rivelatore scintillante economico, equipaggiato con dei fotosensori SiPM. Ciascuno di questi sensori viene replicato al fine di costituire una griglia attorno ad ogni fusto di rifiuti: è possibile posizionare facilmente diversi sensori di radiazione attorno ad ogni fusto in varie posizioni, registrando continuamente l'attività misurata al fine di conoscere istantaneamente il tasso di radiazione e la sua storia([12][13]).

[15] Il Silicon Photomultiplier è un dispositivo sensibile al passaggio dei singoli fotoni costituito da un fotodiodo a valanga costruito su un substrato di silicio.

16 In elettronica digitale il Field Programmable Gate Array è un circuito integrato programmabile il cui comportamento viene definito via software dall'utente finale.

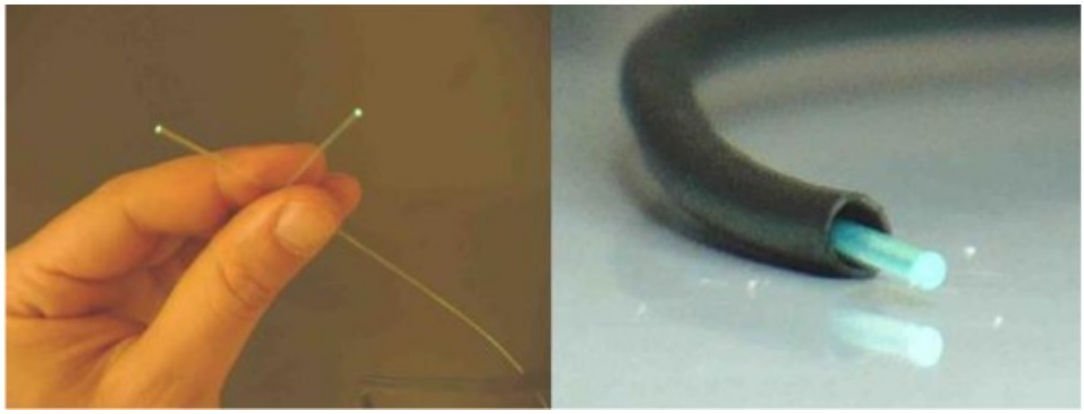


*Fig. 7 : Una fibra scintillante. A destra è mostrata all'interno di una guaina nera che serve come protezione meccanica.*

### 4.3.1 Meccanismo di produzione della luce

Le particelle ionizzanti che attraversano il corpo della fibra scintillante interagiscono con la molecola di PS (vedi figura 6) producono luce, in particolare la molecola di benzene contenuta, per la sua configurazione elettronica, si presta ottimamente all'assorbimento di energia. La molecola di benzene contiene una coppia di elettroni nel livello π più esterno ($S_0$), e numerosi altri livelli eccitati $S_{ij}$ (vedi figura 7). Per interazione con particelle ionizzanti la molecola di benzene viene eccitata, il diseccitamento fino allo stato $S_1$ avviene in maniera non radiativa, mentre il diseccitamento dallo stato $S_1$ allo stato $S_0$, che avviene in tempi inferiori al nanosecondo (fluorescenza), produce fotoni. Fra gli stati $S_1$ ed $S_0$ c'è un livello intermedio $T_1$ (stato di tripletto) ma la transizione $T_1 \rightarrow S_0$ avviene in tempi maggiori (fosforescenza). L'esistenza di numerosi stati roto-vibrazionali determina uno spettro non monocromatico.

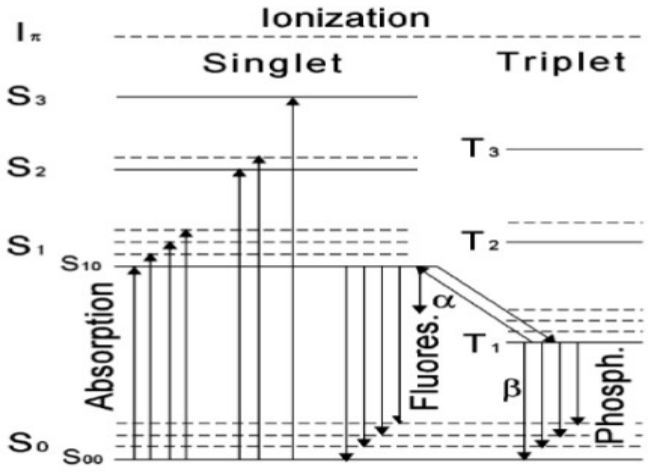



*Fig. 8 : Livelli energetici di una fibra scintillante.*

Altri fenomeni che avvengono normalmente nelle interazioni con particelle ionizzanti sono: ionizzazione degli elettroni dello stato π e l'eccitazione o ionizzazione degli elettroni interni.

Questi ultimi fenomeni concorrono fortemente all'assorbimento di energia ma normalmente non partecipano alla produzione di luce fluorescente, col risultato che solo il 3% dell'energia assorbita viene riemessa radiattivamente [12], e come si può capire questa è una percentuale molto bassa. Un ulteriore problema connesso alla luce prodotta dal PS è che lo spettro ha un massimo per una lunghezza d'onda λ ~330 nm, al limite dello spettro di assorbimento dei fotomoltiplicatori utilizzati. L'aggiunta di un dopante (in una concentrazione del 2%) può risolvere il problema. L'elemento usato come dopante deve avere lo spettro di assorbimento sovrapposto allo spettro di emissione del PS ed un'efficienza quantica, che per un fotomoltiplicatore è definita come il numero di elettroni prodotti per fotone incidente, più alta possibile. Questo secondo materiale fluorescente converte la luce ultravioletta emessa dal PS in luce visibile (~420 nm) e riduce cosi anche il fenomeno dell'auto-assorbimento.

In questo modo dopo una distanza dell'ordine di 500 $\mu$m dal punto in cui la particella ionizzante ha interagito nella fibra tutti i fotoni appartengono allo spettro di emissione della molecola dopante e soprattutto a quella di assorbimento del fotomoltiplicatore. Si ottiene in tal modo un "traslatore" di lunghezza d'onda (*wavelength shifter*).

Le caratteristiche generali di una fibra scintillante, utilizzata nell'apparato sperimentale oggetto della presente tesi, sono riportate nella seguente tabella.

| | |
|---|---|
| Produttore | Pol.Hi.Tec |
| Modello | BC-408 |
| Forma | cylindrical |
| Lunghezza | 120 cm |
| Diametro | 1 mm |
| Base | Polyvinyltoluene |
| Densità | 1.032 g/cc |
| Indice di rifrazione | 1.58 |
| Lunghezza d'onda di emissione massima | 425 nm |
| Lunghezza di attenuazione della luce ($\lambda_{att}$) | 210 cm |
| Resa di scintillazione | 10 photons/keV |
| n° atomi di H per $cm^3$ ($x10^{22}$) | 5.23 |
| n° atomi di C per $cm^3$ ($x10^{22}$) | 4.74 |
| Rapporto (H:C) | 1.104 |

*Fig. 9 : Proprietà della fibra scintillante attualmente utilizzata nel progetto DMNR.*

I fotoni prodotti per decadimento degli stati molecolari eccitati dalla radiazione incidente sono emessi isotropicamente. E' possibile calcolare la percentuale di quelli prodotti che raggiungono le estremità della fibra.[13]

### 4.3.2 CARATTERISTICHE DELLA FIBRA SCINTILLANTE

La soluzione adottata dal DMNR e quindi implementata nella nostra applicazione risulta essere semplice, affidabile, gestibile, sensibile ed ha un costo ridotto. Inoltre presenta i seguenti requisiti: validità, bassa efficienza intrinseca, per non essere saturi, ma al stesso tempo la loro efficienza geometrica deve essere flessibile, per esempio, deve essere possibile cambiare la lunghezza e lo spessore del rivelatore.

Per dimostrare l'attendibilità della fibra scintillante adoperata sono stati eseguiti dei test, delle misure di precisione e delle simulazioni: il sensore, costituito dalla fibra e da due SiPM alle sue estremità, è stato sottoposto ad una sorgente con bassa attività come il $CO^{60}$ (cobalto- 60) e $CS^{137}$ (Cesio-137). La sorgente è stata posta a contatto con la fibra in tre diverse posizioni, come vedremo nel capitolo simulazione.

### 4.3.3 VANTAGGI E SVANTAGGI

Le fibre scintillanti, come rivelatori di radiazione, presentano diversi vantaggi quali:

• *esse danno una risposta rapid*a, permettendo il loro uso anche in rivelatori in cui è prevista un'alta frequenza di eventi;

• esse *hanno una lunghezza di attenuazione* di qualche metro: ciò da la possibilità di costruire rivelatori di adeguate dimensioni o comunque di portare il segnale lontano dall'apparato;

• esse possono essere *usate in rivelatori* magnetizzati: in tal caso i fenomeni di produzione di luce non vengono modificati.

# 5 PROTOTIPO DI SIMULAZIONE

In questo capitolo sono riportati i dettagli del codice sviluppato. La prima osservazione da effettuare consiste nel metodo di funzionamento del software. Geant4 gestisce le chiamate ai metodi con il modello ad *eventi*. Ovvero, ogni volta che uno step della simulazione genera un evento, vengono richiamati i metodi ad esso associati. Il compito del programmatore è quindi quello di decidere se catturare o meno l'evento, implementando il codice necessario per la sua gestione. Come già descritto brevemente nel Capitolo 3, Geant4 gestisce due tipi di classi, quelle obbligatorie *(mandatory user classes)* e quelle opzionali *(optional user classes)*.
Prima di iniziare con l'analisi dettagliate delle classi implementate per la realizzazione di una geometria di monitoraggio per una sorgente radioattiva di $^{137}$Cs, dobbiamo fare un breve cenno sul metodo *main*, nel quale devono essere definiti alcuni *oggetti manager* che consentono l'assemblaggio degli "strumenti" occorrenti al simulatore e contenuti nelle classi.

## 5.1 Il main del programma

Geant4 non fornisce un metodo *main()* ma occorre implementarlo sulla base delle proprie esigenze. Il main di qualsiasi programma Geant4 è pressoché identico, in quanto tutte le operazioni per la gestione della simulazione vanno effettuate dentro le classe. Diamo quindi una rapida visione di insieme delle varie porzioni di codice che compongono il main().
Innanzitutto occorre creare le istanze delle due classi manager che gestiscono l'inizializzazione dell'applicazione: *G4RunManager* e *G4UIManager*.

- *G4RunManager* è la classe manager preposta al controllo del flusso della simulazione attraverso la gestione del susseguirsi degli eventi all'interno di ciascun *run* ed è inoltre responsabile dell'inizializzazione dei parametri della simulazione. A questo scopo occorre fornire al *runManager* le informazioni necessarie per configurare i run, cioè: la *struttura del detector* (in termini di

geometria, materiali e sensibilità di rivelazione), le *particelle* e i *processi fisici* da simulare, il *setup* dell'evento primario. Quanto appena detto, è definito nel codice sottostante mediante l'inizializzazione *runManager e* delle mandatory user class *DetectorConstruction*, *PhysicsList* e *PrimaryGeneratorAction.*

.

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

G4VUserDetectorConstruction* detector = new DetectorConstruction();
runManager->SetUserInitialization(detector);
G4VUserPhysicsList* physics = new PhysicsList();
runManager->SetUserInitialization(physics);
G4VUserPrimaryGeneratorAction*gen_action = new PrimaryGeneratorAction();
runManager->SetUserAction(gen_action);
```

*Fig. 10: Creazione dell'istanza G4RunManager e inizializzazione delle mandatory user classes*

Le informazioni relative al generatore di particelle, racchiuse nella user action class *PrimaryGeneratorAction* vengono passate al *runManager* attraverso il metodo *SetUserAction()* come in Fig. 10. Lo stesso vale per tutte le altre *user action classes*; in particolare per l'applicazione è stata istanziata e inizializzata la classe *RunAction*, derivata dalla classe astratta *G4UserRunAction* del toolkit.

- *G4UIManager* è la classe manager della categoria *intercoms,* categoria definita nel secondo capitolo, che fornisce un interprete dei comandi espandibile e consente all'utente di interagire con tutte le categorie.

Un'applicazione Geant4 può essere eseguita in modalità *"hard coded" bacth mode* ovvero tramite l'uso diretto delle classi di Geant4, definendo i passi per eseguire la simulazione direttamente nel codice *C++,* che però necessita di ricompilazione ogni qualvolta si cambiano i parametri di simulazione tipo numero di eventi generati, oppure in *"bacth mode"* che permette l'esecuzione leggendo da file di comandi opportunamente inizializzati definiti ***"macro file"***.

L'implementazione del metodo *main()* per l'applicazione, come mostra il Codice in *Fig.11*, prevede la possibilità di scegliere tra due interfacce testuali rappresentate dalle classi *G4UIterminal e G4UItcsh* (la prima è valida per tutte

le piattaforme supportate da Geant4, la seconda per sistemi Solaris e Linux) sulla base del valore della variabile d'ambiente *G4UI_USE_TSCH*. In alternativa è anche possibile fare eseguire l'applicazione in modalità batch leggendo i comandi da un macro file passato come parametro al metodo main all'avvio (oppure in modalità interattiva attraverso l'uso del comando *control/execute "nomefile.mac"*).

```
        G4UImanager * UImanager = G4UImanager::GetUIpointer();
        if (argc!=1) { // batch mode
        G4String command = "/control/execute ";
        G4String fileName = argv[1];
        UImanager->ApplyCommand(command+fileName);
        }
        else {
        // interactive mode : define UI session
        G4UIsession * session = 0;
        #ifdef G4UI_USE_TCSH
        session = new G4UIterminal(new G4UItcsh);
        #else
        session = new G4UIterminal();
        #endif
        session->SessionStart();
        delete session;
}
```

*Fig.11: Inizializzazione dell'oggetto user interface manager per l'esecuzione del file batch o per la definizione dell'interfaccia utente per l'immissione di comandi da shell*

Per rendere l'applicazione in grado di fornire una visualizzazione del detector e delle traiettorie delle particelle è necessario istanziare un oggetto *Visualization Manager*, istanza della classe *G4VisManager*, come mostrato nel codice della *Fig. 12*.

```
#ifdef G4VIS_USE
G4VisManager* visManager = new G4VisExecutive;
visManager->Initialize();
#endif
```

*Fig.12: Creazione e inizializzazione del Visualization Manager*

L'utilizzo della variabile d'ambiente *G4VIS_USE* consente di costruire facilmente un eseguibile privo di visualizzazione, che permette un'esecuzione e quindi una simulazione piuttosto veloce. Tuttavia la definizione del *Visualization Manager* non implica che debba essere visualizzata necessariamente la grafica: essa potrà essere avviata o meno in fase di

esecuzione attraverso i comandi interattivi oppure, ancora una volta, in modo batch con i macro file.

La seguente schermata mostra il risultato della simulazione mediante *Visualization Manager* utilizzando un opportuno file di macro *macro.mac* definito sotto:

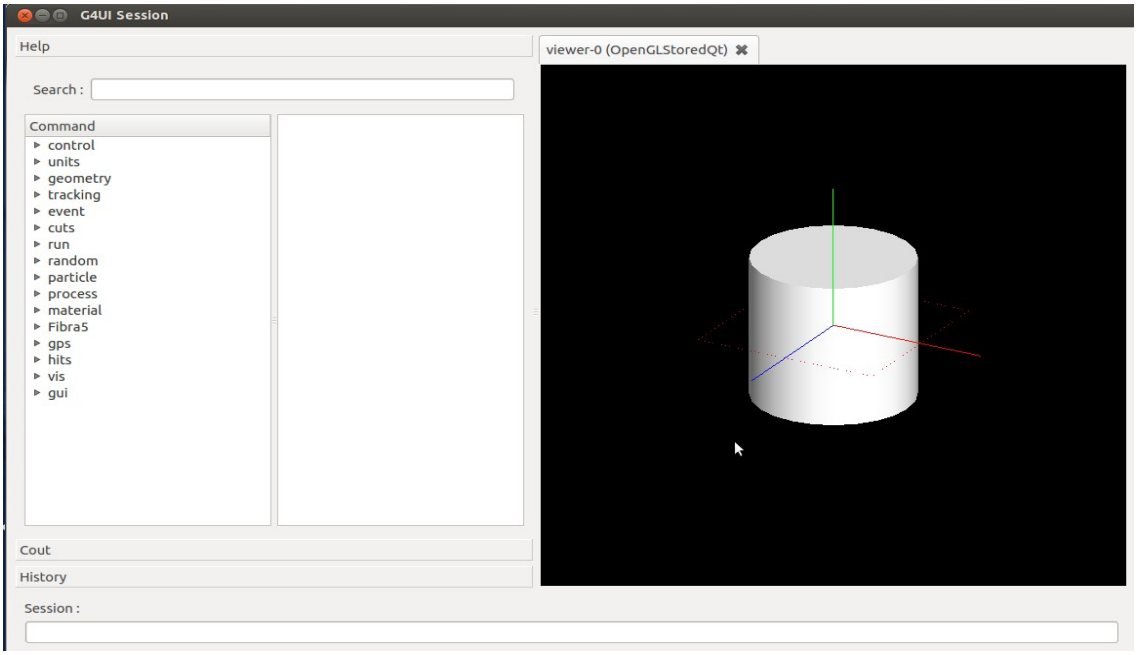



*Fig. 13: GUI della simulazione.*

```
#Use this open statement to create an OpenGL view:
/vis/open OGL 600x600-0+0
/vis/drawVolume
/vis/viewer/set/style surface
/vis/viewer/set/viewpointThetaPhi 0 0
/vis/viewer/zoomTo 1
/vis/scene/add/axes 0 0 0
/control/verbose 2
/tracking/verbose 0
/run/verbose 2
/event/verbose 0
```

*Fig. 14: macrofile per avviare la simulazione.*

Durante la creazione del nostro software di simulazione è stato indispensabile l'utilizzo della visualizzazione grafica esclusivamente in fase di sviluppo della geometria, in modo da verificare la corrispondenza tra il dispositivo creato e quello progettato. Invece durante le vere simulazioni, la modalità grafica è

stata disattivata in modo da aggravare il meno possibile sul carico di lavoro alla CPU, con il conseguente abbreviamento dei tempi di simulazione.

## *5.2 CLASSI IMPLEMENTATE*

Come già discusso nel terzo capitolo Geant4 prevede l'implementazione di alcune classi obbligatorie, denominate *mandatory user classes*, fornite dal toolkit come classi astratte da derivare per la definizione degli elementi base della simulazione: la geometria, i processi fisici e il generatore di particelle. Le altre user classes messe a disposizione dal toolkit sono invece opzionali e riguardano la definizione degli elementi per interagire con il simulatore nei vari stadi della simulazione. Le mandatory user classes dell'applicazione sono: *la DetectorConstruction*, la *PhysicsList* e la *PrimaryGeneratorAction.*

Nel seguito saranno illustrati i dettagli che riguardano queste tre classi obbligatorie e le optional user classes[17] definite per la gestione della simulazione, per implementare la risposta del sensitive detector, per la gestione della *hit collection* e per la definizione di comandi aggiuntivi per impostare a *runtime* alcuni parametri per la simulazione.

[17] Rif 3, pag 30.

### 5.2.1 Detector Construction

La classe *DetectorConstruction* è la prima classe obbligatoria e deve sempre essere implementata. Contiene la definizione della geometria del detector e del fusto di contenimento, cioè le dimensioni, i materiali di cui sono composti, la posizione e le zone sensibili, nel caso del detector *(Sensitive Detector*). Essa deriva la *mandatory user class G4VUserDetectorConstruction* del codice Geant4 che contiene il metodo virtuale *Construct()* che deve essere implementato nella *DetectorConstruction.cc* e che sarà invocato dal *runManager* durante l'inizializzazione dell'applicazione. All'interno di questo metodo deve essere inserito il codice per la definizione della geometria nonché del volume "world". Il metodo dovrà restituire il puntatore all'istanza del volume fisico che rappresenta l'intero rivelatore. Il codice di Fig.15 riporta il contenuto del file *DetectorConstruction.hh* che contiene la dichiarazione dei metodi e degli attributi della classe.

```
class DetectorConstruction : public G4VUserDetectorConstruction
{
	public:
	// Constructor
		DetectorConstruction();
	// Destructor
		~DetectorConstruction();
	public:
	// Construct geometry of the setup
		G4VPhysicalVolume* Construct();
	private:
	// define needed materials
		void DefineMaterials();
	// initialize geometry parameters
		void ComputeParameters();
	// Construct geometry of the detector
		G4VPhysicalVolume* ConstructFibra();
		G4VPhysicalVolume* ConstructTubs();
	private:
	// Materials
		G4Material* air;
		G4Material* polystyrene;
		G4Material* plexiglass;
		G4Material* vacuum;
		G4Material* concrete;
	// global mother volume
		G4LogicalVolume * logicWorld;
		G4double halfWorldLength;
	// Detector Geometry
		G4VPhysicalVolume* physiFibraCladding;
		….....
		G4VPhysicalVolume* TubsLogic;
		G4VPhysicalVolume* TubsLogic2;
		G4VPhysicalVolume* physiFibraCore;
		….....
		G4VPhysicalVolume* TubsCore;
		….....
	// Parameters for detector
		G4double fibraLen;
		G4double fibraRadius;
		G4ThreeVector posFibra;
		G4double TubsLen;
```

*Fig. 15: DetectorConstruction.hh, attributi e metodi della classe DetectorContruction*

Il *DetectorConstruction* contiene, come abbiamo appena detto la definizione della geometria dell'applicazione sviluppata, partendo dal *volume madre* che è il contenitore principale all'interno del quale verranno posizionati i volumi del dispositivo da realizzare. Ci occupiamo ora del settaggio dei materiali e delle proprietà della geometria da realizzare che vengono definiti all'interno di questa classe.

### 5.2.1.1 Definizione del fusto di contenimento

Il fusto è posizionato nell'origine del sistema di riferimento all'interno del *volume madre*, è costituito da un contenitore cilindrico[18]. Nel metodo *ComputeParameters()* della classe *DetectorConstruction,* riportato in figura, sono definite le dimensioni del volume world e del fusto di contenimento.

```
void DetectorConstruction::ComputeParameters()
{
// This function defines the defaults  of the geometry construction
  // ** world **
  halfWorldLength = 10.0*m;
 //** SOURCE **
  TubsLen = 0.5*m;
 }
```

*Fig. 16: DetectorConstruction.cc, metodo ComputeParameters()*

Di seguito è riportato il codice del metodo *ConstructTubs(),* per la costruzione del fusto, utilizzando la classe G4Tubs e riempiendolo con il materiale concrete (cemento) definito nel metodo *DefineMaterials()*.

```
G4VPhysicalVolume* DetectorConstruction::ConstructTubs()
{           G4RotationMatrix* pR = new G4RotationMatrix();
            pR->rotateX(90.*deg);
            G4double TubsRaggioEst = 0.5*m;
      G4Tubs* TubsSolid = new G4Tubs("TubsSolid", //its name
                                      0,                    //inner radius
                                    TubsRaggioEst,          //outer radius
                                      TubsLen,  //half length in z
                                      0,                    //Starting angle in phi CLHEP::twopi);
            // 2nd: define logical volume----------------------------------------------
            G4LogicalVolume* TubsLogic = new G4LogicalVolume( TubsSolid,          //its solid
                                                                polystyrene,        //its material
                                                               "TubsLogic");        //its name
            // 3rd: define the physical volume -----------------------------------
            TubsCore = new G4PVPlacement( pR,       //no rotation
                                          G4ThreeVector(0,0,0),         //translation
                                          TubsLogic,                    //its logical volume
                                          "TubsCore",                   //its name
                                          logicWorld,                   //its mother volume
                                          false,
                                          1);          }
```

*Fig. 17: metodo ConstructorTubs()per la costruzione del fusto.*

[18] Rif. Cap. 3, pag 29

In Figura 18 è mostrato quanto detto sopra in due differenti angoli visuali.

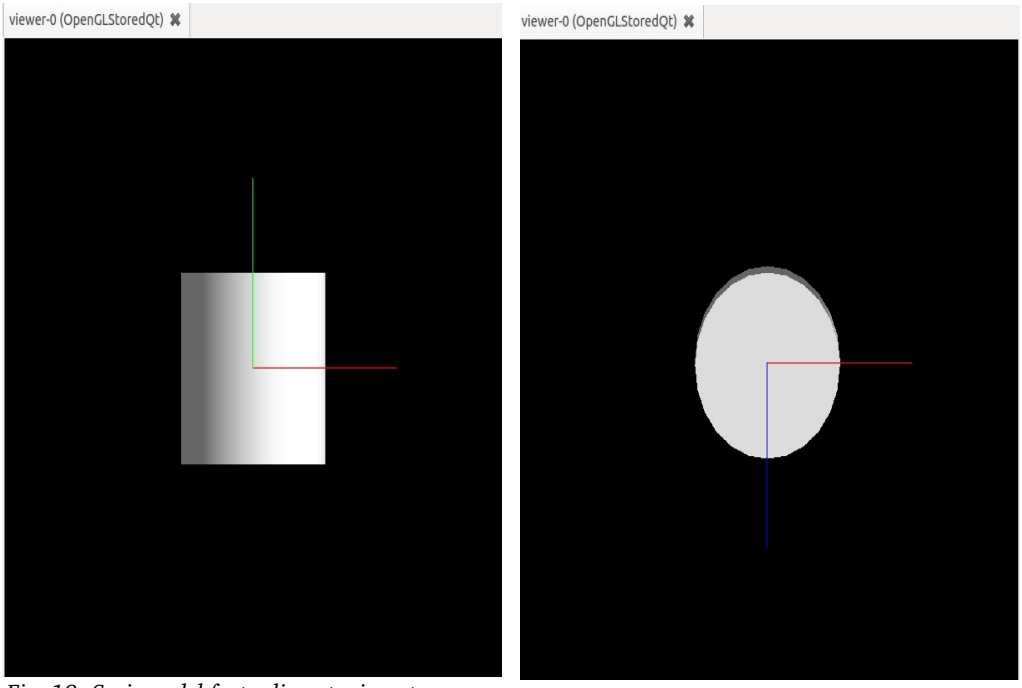



*Fig. 18: Sezione del fusto di contenimento.*

### 5.2.1.2 Definizione del rivelatore

Il rivelatore è costituito da quattro fibre[19]. Ciascuna fibra è composta da un *cladding* di plexiglass e da un *core* di polistirene. Questa classe contiene, oltre alla definizione delle proprietà geometriche, il codice per la definizione del volume sensibile, ovvero l'oggetto sensitive detector della classe *FibraSensitiveDetector* descritta più avanti, che serve a definire le parti del volume sensibili al passaggio delle particelle. Come visto per il fusto di contenimento, anche qui il metodo *ComputeParameters()* della classe DetectorConstruction definisce le dimensioni dei volumi che compongono il detector.

```
void DetectorConstruction::ComputeParameters()
{
// This function defines the defaults // of the geometry construction
// ** Fibra **
fibraLen = 1200*mm;
fibraRadius = 0.5*mm;
posFibra = G4ThreeVector(0,0,0);
```

*Fig. 19: DetectorConstruction.cc, metodo ComputeParameters() per la definizione della fibra*

[19] Rif. Cap.3, pag 29

Di seguito è riportato il codice del metodo *ConstructFibra()*: prima di tutto è stato definito il *core* della fibra, utilizzando la classe *G4Tubs* e riempiendolo con il materiale *polystyrene* definito nel metodo *DefineMaterials()*. Successivamente sono stati istanziati gli oggetti necessari per costruire il *cladding* costituito da *plexiglass*, anch'esso definito nel metodo *DefineMaterials()*. Dopo aver definito i volumi, per rendere il core della fibra sensibile al passaggio delle particelle, si definisce il *sensitive detector*. Si crea l'oggetto *FibraSensitiveDetector* e si registra tramite l'oggetto gestore *SDMan (G4SDManager)* attraverso il metodo *AddNewDetector()*. Successivamente si invoca il metodo *SetSensitiveDetector()* del corrispondente volume logico che si vuole rendere sensibile (in questo caso il core), passandogli come parametro l'oggetto sensitive precedentemente istanziato e registrato tramite il *Sensitive Detector Manager*. Nel nostro caso specifico sono state definite le geometrie di quattro fibre posizionate attorno al fusto e ruotate appositamente mediante un opportuno uso della Matrice di Rotazione. Le rotazioni utilizzate sono mostrate nel codice sottostante, definite all'inizio della classe DetectorConstruction e richiamate nella definizione della fibra opportuna.

```
G4RotationMatrix* pRot = new G4RotationMatrix();
pRot->rotateY(90.*deg);
G4RotationMatrix* pRot2 = new G4RotationMatrix();
pRot2->rotateX(0.*deg);
G4RotationMatrix* pRot3 = new G4RotationMatrix();
…….....
```

*Fig. 20: Dichiarazione delle rotazioni utili per il posizionamento della fibra*

```
G4VPhysicalVolume* DetectorConstruction::ConstructFibra()
{
G4double halfFibraLen = fibraLen/2;
/* FIBRA */
// *** CLADDING ***
        // 1st oggetto solido
        G4double claddingRaggioInt = fibraRadius – 2*fibraRadius*0.03;
        G4double claddingRaggioEst = fibraRadius;
        G4Tubs* fibraCladdingSolid = new G4Tubs( "fibraCladdingSolid",   //its name
                                                 claddingRaggioInt,    //inner radius
                                                 claddingRaggioEst,          //outer radius
                                                 halfFibraLen,               //half length in z
                                                 0,                          //Starting angle in phi
                                                 CLHEP::twopi);              //Ending angle in phi
        // 2nd: definire l'oggetto logico
        G4LogicalVolume* fibraCladdingLogic = new G4LogicalVolume( fibraCladdingSolid,
                                                 //its solid
                                                 plexiglass,                 //its material
                                                 "fibraCladdingLogic");      //its name
        // 3rd: definire l'oggetto fisico
        //physiFibraCladding = new G4PVPlacement(0,       //no rotation
        physiFibraCladding = new G4PVPlacement( pRot,    // rotazione
                                                 posFibra, //translation
                                                 fibraCladdingLogic,  //its logical volume
                                                 "FibraCladding",       //its name
                                                 ogicWorld,                  //its mother volume
                                                 false,
                                                 0);          //copy number

// *** CORE *** FIBRA1
        // 1st oggetto solido----------------------------------------------------
        G4double coreFibraRaggioEst = fibraRadius - 2*fibraRadius*0.03;
        //Create the fiber volume: a tubs
        G4Tubs* coreFibraSolid = new G4Tubs("coreFibraSolid",         //its name
                                                 0,                          //inner radius
                                                 coreFibraRaggioEst, //outer radius
                                                 halfFibraLen,               //half length in z
                                                 0,                          //Starting angle in phi
                                                 CLHEP::twopi);          //Ending angle.....
        // 2nd: define logical volume-------------------------------------------
        G4LogicalVolume* coreFibraLogic = new G4LogicalVolume( coreFibraSolid,          //its solid
                                                                polystyrene,            //its material
                                                                "coreFibraLogic"); //its name
        // 3rd: define the physical volume--------------------------------
                physiFibraCore = new G4PVPlacement( pRot,          //no rotation
                                                 G4ThreeVector(0,0,-590),    //translation
                                                 coreFibraLogic,             //its logical volume
                                                 "physiFibraCore",     //its name
                                                 logicWorld,                 //its mother volume
                                                 false,
                                                 1);                         //copy number
// create a SD
FibraSensitiveDetector* sensitive = new FibraSensitiveDetector("Fibra");
// add it to the SD manager
G4SDManager* sdman = G4SDManager::GetSDMpointer();
sdman->AddNewDetector(sensitive);
// aggiungo il SD al volume logico
coreFibraLogic->SetSensitiveDetector(sensitive);
return physiFibraCladding;
}
```

*Fig. 21: Metodo ConstructorFibra()per la costruzione e la sensibilizzazione della fibra.*

Di seguito è mostrata il risultato della geometria definita sopra.

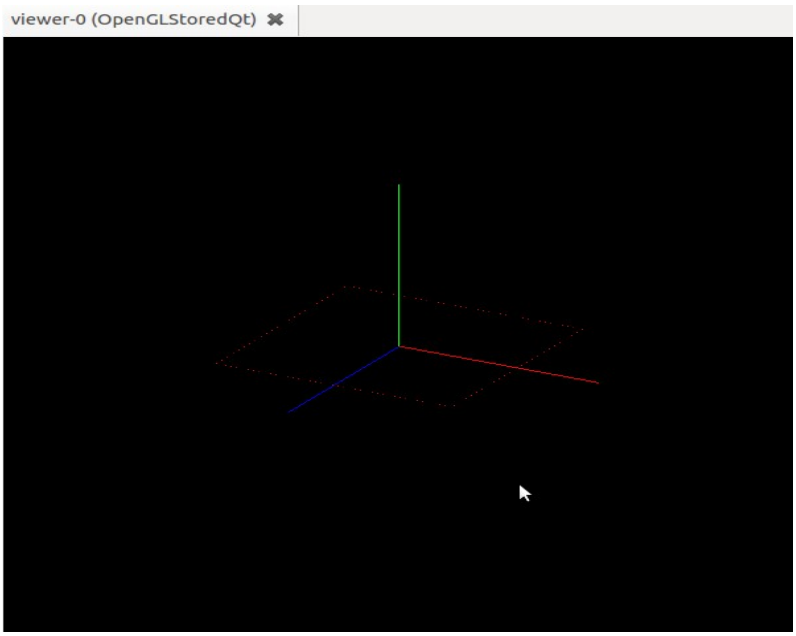



*Fig. 22: Disposizione delle quattro fibre.*

#### 5.2.1.3 Materiali definiti

Il metodo *DefineMaterials()* contenuto nella classe DetectorConstruction definisce i materiali componenti il rivelatore, il fusto e l'ambiente circostante. Ho definito il materiale aria (Air) e vuoto(vacuum) per il volume world, polistirene (polistyrene) e plexiglass per la fibra, e cemento(concrete) per il fusto accedendo ad un database di materiali già definito[20].

```
Void DetectorConstruction::DefineMaterials()
{
        //Get Materials from NIST database
        G4NistManager* man = G4NistManager::Instance();
        man->SetVerbose(0);
        // define NIST materials
        polystyrene= man->FindOrBuildMaterial("G4_POLYSTYRENE");
        plexiglass = man->FindOrBuildMaterial("G4_PLEXIGLASS");
        air = man->FindOrBuildMaterial("G4_AIR");
        vacuum = man->FindOrBuildMaterial("G4_Galactic");
        concrete = man->FindOrBuildMaterial("G4_Concrete);
}
```

*Fig. 23: Materiali utilizzati.*

[20] Rif. Cap 3 pag 30

### 5.2.2 Physics List

Nella classe *PhysicsList* che deriva la mandatory user class *G4VUserPhysicsList* sono definiti i processi fisici e le particelle che intervengono nelle simulazioni. Questa classe contiene tre metodi virtuali da implementare: *ConstructProcess(), ConstructParticle() e SetCuts()*. I primi due devono contenere la definizione dei processi fisici e delle particelle, il terzo serve per impostare i valori di cut-off per le particelle definite. Ecco in dettaglio l'implementazione dei tre metodi:

```
class PhysicsList: public G4VUserPhysicsList
{
public:
  //!Constructor
            PhysicsList();
  //! Destructor
            ~PhysicsList();
   protected:
    //! Construct particles
            void ConstructParticle();
    //! Construct physics processes
            void ConstructProcess();
    //! Define user cuts
            void SetCuts();
            void ConstructEM();
     private:
             G4VPhysicsConstructor*  emPhysicsList;
            G4VPhysicsConstructor* heHadronIon;
            G4VPhysicsConstructor* hiPion;
            G4VPhysicsConstructor* hiIon;
            G4VPhysicsConstructor* hiProtonNeutron;
            G4VPhysicsConstructor* particles;
};
```

*Fig. 24 : PhisycList.hh attributi e metodi della classe PhisycList*

Geant4 fornisce vari tipi di particelle, ciascuna rappresentata da una classe derivata dall'interfaccia G4ParticleDefinition. Tali particelle sono suddivise in 6 maggiori categorie: *lepton, meson, baryon, boson, shortlived e ion*. A ciascuna categoria corrisponde una classe finalizzata a mantenere il codice più ordinato mediante il raggruppamento della definizione di tutte le particelle relative a quella categoria in una sola riga di codice[15]. É, però, anche possibile definire singolarmente le particelle, che devono essere istanze singleton, come mostrato sotto.

```
void PhysicsList::ConstructParticle()
{
// pseudo-particles
        G4Geantino::Geantino();
        G4ChargedGeantino::ChargedGeantino();
// particles
        G4Electron::Electron();
        G4Positron::Positron();
        G4Proton::Proton();
// GAMMA
        G4Gamma::Gamma();
}
```

*Fig. 25: Metodo ConstructParticle() della classe PhysicsList contenente la definizione delle particelle.*

Le particelle abilitate per la simulazione sono: gamma, elettroni, positroni e protoni. Il Geantino rappresenta una pseudo particella usata perlopiù per eseguire delle verifiche sulla correttezza geometrica del sistema e non interviene ai fini dell'esito delle simulazioni.

Per una descrizione più dettagliata dei metodi e delle particelle dei processi fisici si rimanda al *Geant4 User's Guide for Application Developer* [18] e al *Software Reference Manual* [19].

### 5.2.3 Primary Generator Action

Questa classe è implementata derivando la mandatory user class *G4VUserPrimaryGeneratorAction* del codice Geant4 e contiene tutto ciò che riguarda il setup dell'evento primario, cioè il tipo di particella che deve essere generata, la posizione, il momento, l'energia, etc.

La *G4VUserPrimaryGeneratorAction* contiene il metodo virtuale *GeneratePrimaries()*, invocato all'inizio di ogni evento, implementato per la generazione dell'evento primario ed avviato attraverso il metodo *generatePrimaryVertex()*. Le caratteristiche del *PrimaryVertex()* sono racchiuse nei due oggetti *G4ParticleGu*n[16] e *G4GeneralParticleSource*[20] che sono l'implementazione vera e propria del metodo *G4PrimaryGeneretor*.

Il *G4ParticleGu*n permette di generare una particella primaria di una certa energia da un certo punto a certo tempo ad una certa direzione, invece, G4GeneralParticleSource permette di generare le primary vertex a caso

sulla superficie di un qualunque volume, la direzione di moto e l'energia cinetica può anche essere random. La distribuzione(isotropica, SolidAngle...) può essere impostata dai comandi dell'interfaccia utente.
La nostre simulazioni sono state realizzate utilizzando una sorgente puntiforme ad emissione isotropica e una sorgente estesa opportunamente implementata. Mostriamo in dettaglio i metodi della classe *Primary Generator Action.*

```
class PrimaryGeneratorAction : public G4VUserPrimaryGeneratorAction
{
        public:
                G4ParticleDefinition *particle;
                // constructor
                PrimaryGeneratorAction();
                // destructor
                ~PrimaryGeneratorAction();
                // defines primary particles (mandatory)
                void GeneratePrimaries(G4Event*);
                public:
                G4ThreeVector GenerateIsotropicFlux();
                G4ThreeVector DistributionUniformInSolidAngle();
                // set methods
                void setSource(G4String newValue);
                G4ThreeVector setDistributionType(G4String newValue);
                void setDistanceY(G4double newDistance);

                //G4VPrimaryGenerator* InitializeGPS(); //x gps
                // get methods
                G4String getSourceType();
                G4String getDistributionType();
                G4double getDistanceY();
        private:
                G4ParticleGun* gun;
                PrimaryGeneratorMessenger* gunMessenger;
                G4double distanceY;
                G4String distribType;
                G4ThreeVector direct;
                G4String source;
};
```

*Fig. 26: Metodi e attributi della classe PrimaryGeneratorAction*

### 5.2.3.1 Particle Gun

La sorgente puntiforme è realizzata implementando i metodi della classe *G4VUserPrimaryGeneratorAction* che contiene il metodo virtuale GeneratePrimaries(), invocato all'inizio di ogni evento, implementato per la generazione dell'evento primario ed avviato attraverso il metodo *generatePrimaryVertex()* del G4ParticleGun[21]. Gli attributi relativi alla sorgente (distanceY, distribType, source) sono inizializzati nel costruttore,

e mostrati nel codice sottostante, con dei valori di default che potranno essere cambiati interattivamente attraverso alcuni comandi definiti nella classe *PrimaryGeneratorMessenger*. Vediamo il codice per l'implementazione della della sorgente puntiforme.

```
PrimaryGeneratorAction::PrimaryGeneratorAction()
        {
        // create a messenger for this class
        gunMessenger = new PrimaryGeneratorMessenger(this);
        // create particle gun
        gun = new G4ParticleGun();
        particle = G4ParticleTable::GetParticleTable()->FindParticle("gamma");
        gun->SetParticleDefinition(particle);
                *** default settings:
                sorgente: Cobalto60
                distanza: 1 cm
                distribuzione: isotropica
                distanceY=1.0*cm;
                distribType = "iso";
                setDistanceY(distanceY);
                setSource("Co60");
        }
```

*Fig. 27: Metodo Costruttore della classe PrimaryGeneratorAction.*

Tali caratteristiche devono essere settate attraverso i metodi del *G4ParticleGun* (*setParticleMomentumDirection(), setParticlePosition(), setParticleEnergy(), etc.*) invocati all'interno del metodo *GeneratePrimaries()* prima della chiamata al metodo *generatePrimaryVertex()*. La classe contiene i due metodi *DistributionUniformInSolid*Angle() e *GenerateIsotropicFlux()* che restituiscono un oggetto *G4ThreeVector* da passare come parametro al metodo *SetParticleMomentumDirection()* del *particle gun* per settare la distribuzione dei raggi. Quest'ultima può essere rispettivamente isotropica in tutto lo spazio o isotropica nell'angolo solido coperto dal detector. In Figura è riportata una sezione della sorgente isotropica in tutto lo spazio.

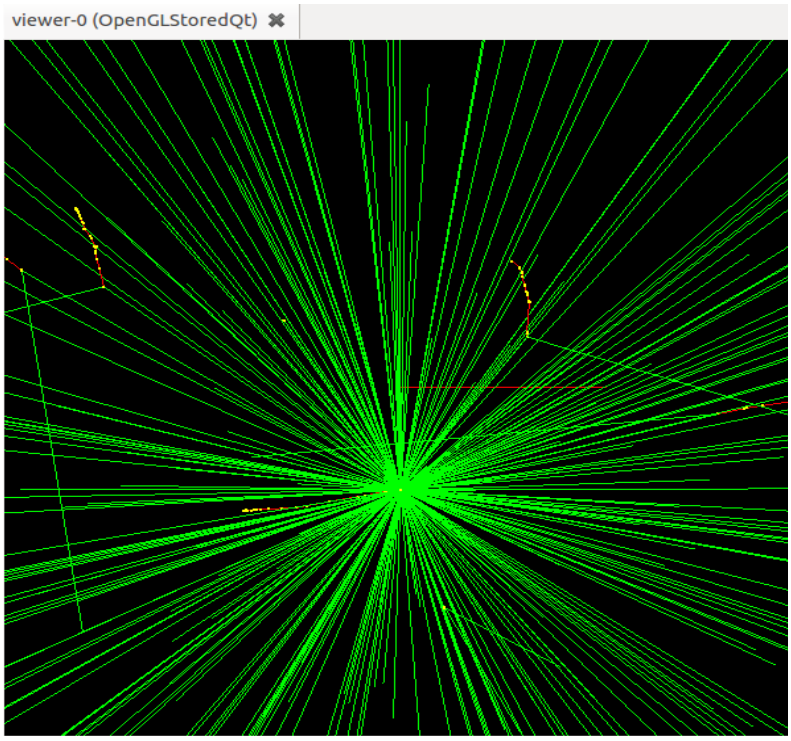



*Fig. 28: Sorgente isotropica su tutto lo spazio.*

#### 5.2.3.2 General Particle Source

La sorgente estesa è realizzata implementando il *G4GeneralParticleSource* che è parte del toolkit Geant4 per le simulazioni ad alta energia di trasporto delle particelle. In particolare, consente le specifiche spettrali, la distribuzione spaziale e angolare delle particelle primarie di origine. Questo può essere fatto tramite riga di comando o macro basato su input. L'utente esperto può ovviamente utilizzare una esecuzione hard-code che utilizzano i metodi e le classi dei GPS che sono descritti in maggiore dettaglio nella nota tecnica [22]. Per consentire una facile definizione delle sorgenti, si presume la matrice di rotazione viene calcolato automaticamente all'interno di G4GeneralParticleSource. I punti di partenza di particelle sono sempre distribuite in modo omogeneo sulle superfici 2D o 3D. L'energia delle particelle del G4GeneralParticleSource può essere impostato per seguire  diverse funzioni; le funzioni disponibili

sono mono-energetico, lineare, esponenziale, legge di potenza, gaussiano, bremsstrahlung, corpo nero e raggi cosmici gamma diffusa. L'utente può influenzare la distribuzione di energia, al fine di velocizzare la simulazione[23].

Nel caso specifico la sorgente estesa è stata realizzata utilizzando un *macro file,* mostrato di sotto*,* che implementa una sorgente di forma cilindrica al fine di poter simulare in maniera quanto più reale possibile le tipiche perdite all'interno del fusto.

```
#/gps/particle gamma
#/gps/pos/type Volume
#/gps/pos/shape Cylinder
#/gps/pos/centre 0. 0. 0. cm
#/gps/pos/radius 0.5 m
#/gps/pos/halfx 0.5 m
#/gps/pos/halfy 0.5 m
#/gps/pos/halfz 0.5 m
#/gps/ang/type iso
#/run/beamOn 100000
```

*Fig.29: Macro file utilizzato per l'implementazione di una sorgente estesa di forma cilindrica.*

In Figura è riportata una sezione della sorgente estesa isotropica in tutto lo con un fascio di raggi gamma.

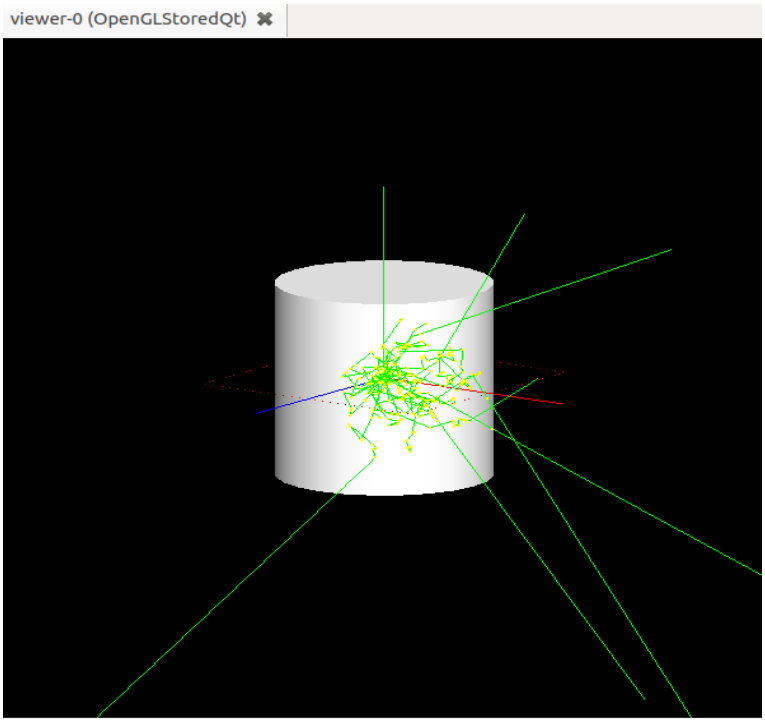



*Fig. 30: Sorgente estesa isotropica su tutto lo spazio.*

### 5.2.4 Fibra Sensitive Detector

Questa classe è stata implementata per definire la risposta del rivelatore. Contiene metodi per gestire una collezione di oggetti, detti *hit*, ognuno dei quali rappresenta un'istantanea dell'interazione fisica di una traccia nella regione sensibile del detector. La *FibraSensitiveDetector* deriva la classe astratta *G4VSensitiveDetector* del codice Geant4 e ne ridefinisce i me*todi Initialize(), Process Hits(), EndOfEvent(),* attraverso i quali avviene la gestione degli *hit* e della *hit collection*. Ciascun oggetto di questa classe conterrà le informazioni che si desidera accumulare nel corso della simulazione durante l'interazione delle particelle con il sensitive detector e sarà collezionato all'interno della *hit collection*.

La *hit collection* è definita nella classe *FibraSensitiveDetector e* rappresenta una collezione di oggetti di tipo *FibraHit*. Tale collezione è istanziata sulla base della template class *G4THitsCollection* del codice Geant4 attraverso una procedura standard riportata di sotto.

```
#include "G4THitsCollection.hh"
... ... ...
// Define the "hit collection" using the template class G4THitsCollection:
typedef G4THitsCollection<FibraHit> FibraHitCollection;
```

*Fig. 31: Classe FibraHit: assegnazione del nome FibraHitCollection al tipo di dato G4THitsCollection.*

Questa definizione permette anche di specificare che la collezione conterrà oggetti di tipo *FibraHit*. Fatto questo la creazione della collezione avverrà nella classe *FibraSensitiveDetector* che conterrà il puntatore all'oggetto *hitCollection*, di tipo *FibraHitCollection*.

Ciascuna hit collection deve avere un nome univoco per ogni evento: dopo aver definito il puntatore si definisce la collezione nel metodo costruttore, assegnando un nome, che in questo caso coincide con l'etichetta *"FibraHitCollection"*, e aggiungendolo al vettore di nomi *collectionNam*e, attributo della classe *G4VSensitiveDetector*.

```
FibraSensitiveDetector::FibraSensitiveDetector(G4String Sdname) : G4VSensitiveDetector(SDname)
{
        G4String myCollectionName = "FibraHitCollection";
        collectionName.insert(myCollectionName);
        hitCount = 0;
        eneTotRun = eneTotEvento = 0;
        initSpettro(); }
```

*Fig. 32: Metodo costruttore della classe FibraSensitiveDetector .*

Successivamente nel metodo *Initialize()* viene istanziato l'oggetto vero e proprio, come mostra nella figura sottostante.

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

*Fig. 33: Metodo Initialize() di FibraSensitiveDetector..*

Il costruttore della hit collection riceve due parametri, di cui il primo è il nome assegnato al *sensitive detecto*r e il secondo è il nome assegnato alla *hit collection*. Questo metodo contiene anche la *hit collection*. Ogni oggetto "evento" attraverso il metodo *AddHitsCollection()* di *G4HCofThisEvent* che riceve come parametri l'ID della collezione e il puntatore alla collezione.
La gestione della collezione di hit avverrà attraverso la classe *FibraSensitiveDetector* che selezionerà gli oggetti *hit* da collezionare attraverso il metodo *ProcessHits()*.

### 5.2.5 Primary Generator Messenger

Geant4 fornisce un ampio set di comandi built-in per l'interfaccia utente che possono essere usati interattivamente, attraverso un macro file in modo batch oppure all'interno del codice C++ con il metodo *ApplyCommand()* della classe G4UImanager. Il toolkit offre anche la possibilità di definire nuovi comandi attraverso una classe messenger, ottenuta derivando la

*G4UImessenger* del codice Geant4, classe base che fa da tramite tra l'interfaccia utente e le classi contenenti il codice per l'esecuzione dei comandi[18]. La classe *messenger* sviluppata per l'applicazione è la *PrimaryGeneratorMessenger*, che contiene la definizione degli oggetti e implementa il metodo *SetNewValue( )* per definire il comportamento da associare ai comandi aggiunti. Un comando corrisponde ad un oggetto di una opportuna classe derivata dalla *G4UIcommand* del toolkit. Per raggruppare i comandi in una *directory* è stato istanziato un oggetto della classe *G4UIdirectory* del toolkit (anch'essa derivata dalla *GUIcommand),* per definire la directory *"Fibra"*, all'interno della quale sono stati posti i comandi aggiuntivi implementati. Gli oggetti istanziati per la definizione dei comandi sono: *distrCmd, sourceCmd, sourceType,* a cui corrispondono i comandi: *setDistrib, setDistance, setSourceType,* che servono rispettivamente a settare il tipo di distribuzione delle particelle irradiate, la distanza della sorgente dal rivelatore e il tipo di sorgente. Di seguito è riportato il codice della classe in questione:

```
PrimaryGeneratorMessenger::PrimaryGeneratorMessenger(PrimaryGeneratorAction* gun) :action(gun)
{
        gunDir = new G4UIdirectory("/Fibra/");
        gunDir->SetGuidance("Particle Gun settings");
        distrCmd = new G4UIcmdWithAString("/Fibra/setDistrib",this);
        distrCmd->SetGuidance("Set the particle gun beam distribution type.");
        distrCmd->SetGuidance(" Choice : iso(default), solidAngle");
        distrCmd->SetParameterName("Distribution type",true);
        distrCmd->SetDefaultValue("iso");
        distrCmd->SetCandidates("iso solidAngle");
        distrCmd->AvailableForStates(G4State_PreInit,G4State_Idle);
        sourceCmd = new G4UIcmdWithADoubleAndUnit("/Fibra/setDistance",this);
        sourceCmd->SetGuidance("Set the distance of the particle source
        detector.");
        sourceCmd->SetGuidance(" Choice : Double and Unit");
        sourceCmd->SetParameterName("Distance",true);
        sourceCmd->SetDefaultValue(1.0);
        sourceCmd->SetDefaultUnit("cm");
        sourceCmd->AvailableForStates(G4State_PreInit,G4State_Idle);
        from
        the
        sensitive
        sourceType = new G4UIcmdWithAString("/Fibra/setSourceType",this);
        sourceType->SetGuidance("Set source type.");
        sourceType->SetGuidance(" Choice : Co60(default), Cs137");
        sourceType->SetParameterName("Source type",true);
        sourceType->SetDefaultValue("Co60");
        sourceType->SetCandidates("Co60 Cs137");
        sourceType->AvailableForStates(G4State_PreInit,G4State_Idle);
```

*Fig.34: Metodo costruttore di PrimaryGeneratorMessenger*

### 5.2.6 Run Action

La classe *RunAction* è implementata derivando la classe virtuale *G4UserRunAction* i cui metodi devono essere riscritti per guadagnare il controllo della simulazione a vari stadi[18]. La *G4UserRunAction* contiene alcuni metodi virtuali che saranno invocati dal *G4RunManager* durante il corso di un run:

- *GenerateRun(),* invocato all'inizio del BeamOn, tipicamente utilizzato per settare alcune variabili che possono influire sulla physics table;
- *BeginOfRunAction(),* invocato prima dell'inizio del ciclo di eventi, può essere usato per inizializzare istogrammi;
- *EndOfRunAction(),* invocato alla fine del run, tipicamente contiene il codice per l'analisi del run eseguito.

Nel metodo costruttore avviene l'assegnazione del puntatore all'oggetto *action* della classe *PrimaryGeneratorAction* ricevuto come parametro, che servirà per accedere ai metodi get della classe. Ciò consentirà di ottenere i valori utili al resoconto della simulazione che verrà stampato a video e su file. Nella classe RunAction sviluppata per l'applicazione sono stati ridefiniti i metodi *BeginOfRunAction() e EndOfRunAction()*.

```
void RunAction::BeginOfRunAction(const G4Run* aRun)
{
        G4cout<<"\n\n################ Run "<< aRun->GetRunID()<< " starting.... "<< G4endl;
        G4SDManager* SDman = G4SDManager::GetSDMpointer();
        FibraSensitiveDetector* sd;
        sd=static_cast<FibraSensitiveDetector*>(SDman->FindSensitiveDetector("Fibra",true));
        sd->hitCount = 0;
        sd->eneTotRun = 0;
        sd->initSpettro();
```

*Fig. 35: Metodo BeginOfRunAction() della classe RunAction*

Il codice mostrato sopra riporta il metodo eseguito all'inizio del *run*, in cui avviene l'inizializzazione delle variabili relative al conteggio degli *hit*, l'accumulo di energia del rivelatore, e lo spettro di energia. Queste variabili non sono altro che gli attributi del *sensitive detector*, che possono essere aggiornate ottenendo il puntatore attraverso il *G4SDManager* che contiene la lista dei volumi sensibili definiti: il metodo

*FindSensitiveDetector()* del *G4SDManager* restituisce il puntatore all'oggetto cercato. Successivamente avviene l'assegnazione dei valori alle variabili di interesse che potranno essere accedute tramite questo puntatore.

Il metodo *EndOfRunAction() ,* contenuto in *Fig.36, è* eseguito alla fine di ogni run e contiene il codice per salvare le informazioni relative al run appena eseguito in un file di testo a cui è assegnato il nome *"output.txt"* che, come mostrato nel codice, conterrà un resoconto sull'esito dei run eseguiti. Inoltre, alla fine del metodo viene invocato il metodo *scrivi_su_file()* della classe FibraSensitiveDetector, che si occupa di salvare lo spettro di energia ottenuto dalla simulazione, opportunamente memorizzato nel vettore spettro[], in un file di testo denominato "spettro" seguito dall'ID del run.

```
void RunAction::EndOfRunAction( const G4Run* aRun )
{
G4SDManager* SDman = G4SDManager::GetSDMpointer();
FibraSensitiveDetector* sd;
sd=static_cast<FibraSensitiveDetector*>(SDman->FindSensitiveDetector("Fibra",true));
ofstream outFile;
outFile.open("output.txt",ios::app);
outFile << "\n################ Run "<< aRun->GetRunID() << flush;
outFile <<"\n          > Source Type : " << action->getSourceType() << flush;
outFile <<"\n          > Distribution Type : " << action->getDistributionType() << flush;
outFile <<"\n          > Source Distance : " << G4BestUnit(action->getDistanceY(), "Length") << flush;
outFile << "\n          > Hits count : " << sd->hitCount << flush;
outFile <<"\n          > Energia totale rilasciata : " << G4BestUnit(sd->eneTotRun,"Energy") << flush;
"\n#####################################\n" << flush;

outFile.close();

G4cout <<"\n          > Source Type : " << action->getSourceType() << G4endl;
G4cout <<"          > Distribution Type : " << action->getDistributionType() << G4endl;
G4cout <<"          > Source Distance : " << G4BestUnit(action->getDistanceY(), "Length") << G4endl;
G4cout <<"          > Hits count : " << sd->hitCount << G4endl;
G4cout <<"          > Energia totale rilasciata : " << G4BestUnit(sd->eneTotRun,"Energy") << G4endl;
G4cout <<"\n#####################################\n" << G4endl;

G4String nomeFile = "spettro";
stringstream ss;
ss << aRun->GetRunID();
nomeFile.append(ss.str());
nomeFile.append(".txt");
sd->scrivi_su_file(nomeFile);
}
```

*Fig.36 : Metodo EndOfRunAction() della classe RunAction*

## 5.3 VARIABILI D'AMBIENTE

Dopo aver implementato il codice di tutte le classi necessarie l'eseguibile viene generato attraverso il meccanismo *GNUmake* controllato da alcuni script (*architecture.gmk, binmake.gmk, GNUmakefile, common.gmk, globlib.gmk*) messi a disposizione dalla distribuzione e contenuti nella cartella di installazione di Geant4 (il cui percorso si trova nella variabile d'ambiente *$G4INSTALL/config*). Gli script in questione servono a definire le regole necessarie per la costruzione degli oggetti, delle librerie, degli eseguibili, etc. Il programma viene compilato invocando il comando ***"make"*** avendo collocato i file sorgenti nella directory *$G4WORKDIR*. Prima di lanciare il comando per la compilazione del codice si devono impostare tutte le variabili d'ambiente necessarie eseguendo i seguenti comandi:

```
export G4WORKDIR=/home/maria/Scrivania/SIMULAZIONE/G4WORKDIR/RUN
exportLD_LIBRARY_PATH=/home/maria/Scrivania/SIMULAZIONE/clhep/2.1.0.1/CLHEP/lib/:
                      $LD_LIBRARY_PATH
source /home/maria/Scrivania/SIMULAZIONE/geant/geant4.9.4.p01/env.sh
export PATH=/usr/local/Trolltech/Qt-4.8.2:$PATH
export QTHOME=/usr/local/Trolltech/Qt-4.8.2
export PATH=/usr/local/Trolltech/Qt-4.8.2/bin:$PATH
```

*Fig. 37: Comandi per impostare le variabili d'ambiente prima di compilare o eseguire codice Geant4.*

La variabile d'ambiente *G4WORKDIR* deve contenere il percorso alla directory di lavoro dove sono contenuti i file sorgenti delle classi, per comodità si lancia un file *setupgeant.sh* che permette l'esportazione di tutte le variabili. Come vediamo si esegue l'exporting delle librerie *CLHEP* che viene stabilito durante la fase di installazione del toolkit, si esegue lo script *env.sh* fornito dal toolkit, che imposta tutte le altre variabili d'ambiente necessarie e l'exoprting delle *Qt* che favoriscono lo sviluppo di un'esecuzione interattiva.

Dopo aver impostato le variabili d'ambiente si può compilare il programma invocando il comando *make* che genera il file eseguibile nella directory *$G4WORKDIR/bin/$G4SYSTEM*. [18]

# 6 Validazione della applicazione sviluppata: confronto con i dati sperimentali

In questo capitolo verranno discusse le verifiche effettuate sulla geometria del sistema sviluppato e sull'attendibilità del codice implementato e gli esiti delle simulazioni eseguite, attraverso l'analisi dei dati ottenuti in forma grafica.

## 6.1 VERIFICA ANALITICA

Prima di tutto, prima ancora di iniziare con le simulazioni dobbiamo procedere con la verifica analitica della geometria del sistema. Per tale ragione sono state condotte delle verifiche attraverso delle simulazioni dapprima in modalità grafica, per avere una visualizzazione della geometria implementata e delle traiettorie generate, e in un secondo momento attraverso il confronto numerico tra il valore dell'angolo solido coperto dal detector ottenuto dalle simulazioni e il valore calcolato analiticamente.

Per verificare l'efficienza geometrica è stato calcolato il valore dell'*angolo solido* del rivelatore visto dalla sorgente. Per definizione l'angolo solido[28] non è altro che l'estensione a tre dimensioni del concetto di angolo piano e rappresenta una misura di quanto largo appare l'oggetto osservato da un determinato punto di vista: geometricamente si può dire che esso costituisce una misura della parte di spazio compresa entro un fascio di semirette uscenti dal punto di osservazione, così come l'angolo piano dà una misura della parte di piano compresa tra due semirette uscenti da un punto. Sia dS un elemento di superficie, $u_n$ la sua normale, $dS_0$ la sua funzione ortogonale e $u_r$ il versore del raggio r uscente da un punto di osservazione O. (Fig. 38)

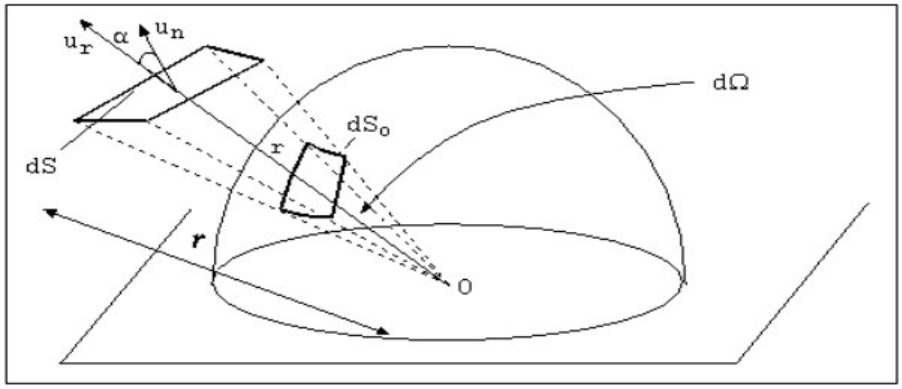



*Fig. 38: Definizione di angolo solido.*

Si definisce angolo solido infinitesimo la quantità

$$d\Omega = \frac{dS\cos\alpha}{r^2} = \frac{dS_0}{r^2}$$

la superficie $dS_0$ rappresenta un elemento di superficie proiettato sulla sfera unitaria. [10] In generale l'angolo solido Ω sotteso da una superficie S è definito come la superficie di una sfera unitaria coperta dalla proiezione della superficie S sulla sfera:

$$\Omega \equiv \int d\Omega = \iint_S \frac{\hat{\boldsymbol{n}} \cdot dS_0}{r^2}$$

$\hat{\boldsymbol{n}}$ è il versore che parte dall'origine del punto di osservazione, $dS_0$ è l'elemento di area infinitesima sulla superficie ed r la distanza dall'origine. In coordinate sferiche (φ latitudine, Θ longitudine) diventa:

$$\Omega \equiv \iint_S \sin\theta \, d\theta \, d\phi$$

L'angolo solido sotto cui un punto interno a una superficie chiusa vede la superficie è sempre 4π che rappresenta il valore massimo dell'angolo solido:

$$\Omega = \iint_0^{2\pi} \sin\theta \, d\theta \, d\phi = 2\pi \int_{\theta_0}^{\theta_1} \sin\theta \, d\theta = 2\pi(\cos\theta_0 - \cos\theta_1)$$

Ponendo e $\Theta_0=0$ e $\Theta_1=\pi$ si ottiene l'angolo solido totale:

$$\Omega = 2\pi(\cos\theta_0 - \cos\theta_1) = 2\pi\big(1-(-1)\big) = 4\pi$$

La frazione di angolo solido vista da una sorgente puntiforme per un rivelatore posto ad una distanza d, avente dimensioni infinitesime (molto piccole rispetto a d), e cioè l'efficienza geometrica $\varepsilon_g$ sarà:

$$\varepsilon_g = \frac{area\ rivelatore}{area\ della\ sfera\ di\ raggio\ d} = \frac{A_{riv}}{4\pi d^2} = \frac{\Omega_{riv}}{\Omega_{tot}}$$

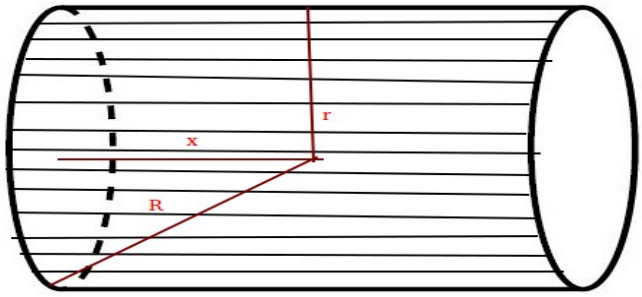
Dove $A_{riv}$ e $\Omega_{riv}$ sono l'area e l'angolo solido del rivelatore e $\Omega_{tot}$ , sarà l'angolo solido totale. Nel caso di un rivelatore di dimensioni finite l'espressione va estesa tramite l'integrale:

$$\Omega = \int d\Omega = \int \frac{dA}{R^2}$$

$\Omega$ è l'angolo solido, $d\Omega$ è l'angolo solido infinitesimo, esprimibile come il rapporto tra l'elemento di superficie infinitesima dA e il valore $R^2$, ovvero il quadrato della distanza dal punto di osservazione. Per ottenere la frazione di angolo solido del rivelatore vista dalla sorgente alla distanza di 1 cm sono state eseguite delle simulazioni, senza visualizzazioni grafiche, allo scopo di sfruttare al massimo le prestazioni per consentire di generare un gran numero di eventi impiegando breve tempo. Le prove sono state eseguite generando le pseudo-particelle (geantini) in maniera isotropica in tutto lo spazio. Il rapporto tra il numero di particelle incidenti nel primo caso e il numero totale di particelle emesse è risultato coerente con il valore atteso per una fibra di lunghezza infinita ottenuto analiticamente. In

realtà il calcolo analitico della porzione di angolo solido è stato semplificato nel seguente modo:

Consideriamo un cilindro costituito da un certo numero *n* di fibre ($n_{fibra}$) costruito a partire dalla stessa fibra, gli eventi contati dalla fibra, cioè $\varepsilon_{geometrico,}$ saranno dati dall'angolo solido della fibra stessa $\Omega_{fibra}$ meno l'angolo solido delle calotte sferiche, $\Omega_{calotte,}$ costruite sulle estremità cilindrica.




Dove x è la lunghezza della fibra ed r è il raggio, rispettivamente pari a 0.60 m e 0.05 cm.

Pertanto i valori analitici sono stati calcolati con i seguenti passi:

$$\Omega_{fibra}= \frac{4\pi-2\Omega calotte}{nfibre}$$

–

- determiniamo il valore dell'angolo solido delle calotte sferiche costruite sul cilindro stesso

$$\Omega_{calotte} = 2\pi( 1 - \frac{x}{R})= 2\pi(0.3) \approx 1.84$$

- determiniamo il numero delle fibre contenute nel cilindro

$$n_{fibre} = \frac{2\pi R}{d} = \frac{2\pi 60}{0.1} = 3768$$

- calcoliamo adesso l'angolo solido della fibra:

$$\Omega_{fibra} = = \frac{4\pi-2\Omega calotte}{nfibre} = = \frac{4\pi-2*1.84}{3768} \approx 0.00236$$

– il valore dell'efficienza geometrica è determinato da:

$$\varepsilon_{geometrico} = \frac{\Omega fibre}{\Omega calotte} = \frac{0.00236}{3768} = 0.000187 = 1.87 * 10^{-4}$$

– quindi il risultato atteso per la nostra implementazione con un valore di 1000000 eventi generati dalla sorgente è:

$$\varepsilon_{geometrico} = (1.87*10^{-4}) * 10^{6} \sim 187$$

– la bontà del risultato analitico è determinato da:

$$\varepsilon_{geometrico} \mp \sqrt{\varepsilon geometrico} = 187 \mp \sqrt{187} = 187 \mp 13$$

## 6.2 VERIFICA DELLE SIMULAZIONI

Per le simulazioni sono state prese in considerazione due tipologie di sorgenti per la generazione degli eventi: una sorgente puntiforme che genera una distribuzione uniforme in tutto lo spazio e una sorgente estesa.

In entrambi i casi verranno lanciati un totale di *1000000 di eventi*, variando nel primo caso la posizione e il numero delle sorgenti rispetto alle fibre, nel secondo caso essendo una distribuzione random di eventi ad ogni *run* varierà il punto di emissione dell'evento stesso. Le simulazioni verranno eseguite sulla seguente geometria:

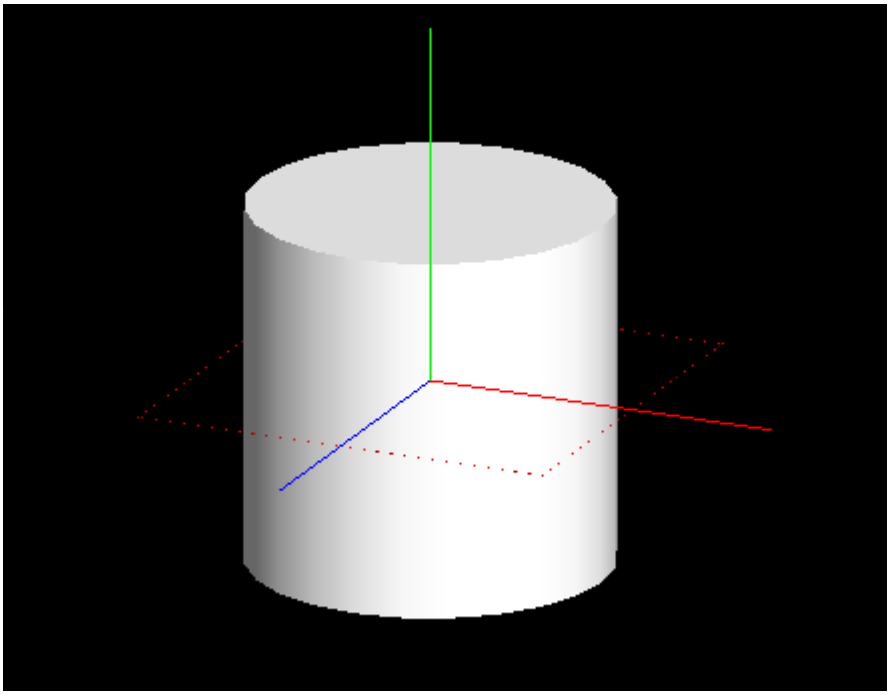


*Fig.31: Geometria del prototipo di monitoraggio.*

Si possono distinguere, in Fig. 31, il *fusto di contenimento*, in grigio, all'interno del quale saranno poste le sorgenti e in rosso(tratteggiato) i dispositivi di conteggio, cioè le *fibre*, disposte attorno al fusto. Verifichiamo l'attendibilità della simulazione realizzato analizzando i risultati generati in un file *output.txt*.

### 6.2.1 RISULTATI DELLA SIMULAZIONE CON SORGENTE PUNTIFORME

La sorgente puntiforme, come abbiamo accennato nei capitoli precedenti, è stata realizzata con l'istanza *G4ParticleGun* della classe *G4VPrimaryGenerator*. Il comando di seguito è utilizzato per generare gli eventi di una sorgente puntiforme:

```
gun = new G4ParticleGun();
particle = G4ParticleTable::GetParticleTable()->FindParticle("gamma");
```

La posizione è determinata dalla funzione:

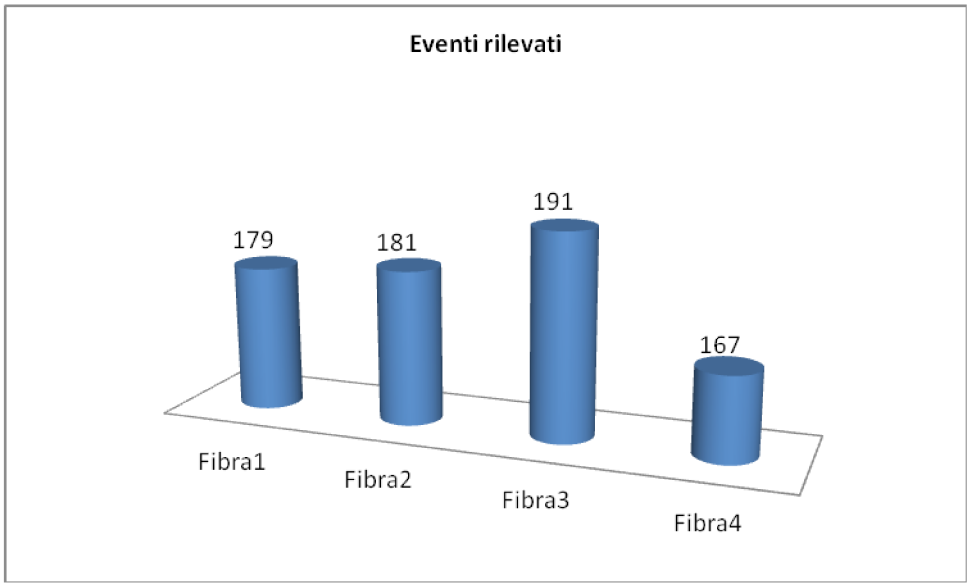
```
distanceY=0.0*mm;
distribType = "iso";
setDistanceY(distanceY);
setSource("Cs137");
```

Per rendere l'esecuzione più veloce si sono utilizzati dei *macro file* contenenti i comandi per l'esecuzione dei run.

Le particelle generate saranno tutte rivelate dal detector a meno di un 6% dovuto alle particelle che colpiscono il cladding (il cui raggio è maggiore di quello del core del 3%). Di seguito sono riportati i file di output con il resoconto delle simulazioni e la loro rappresentazione grafica:

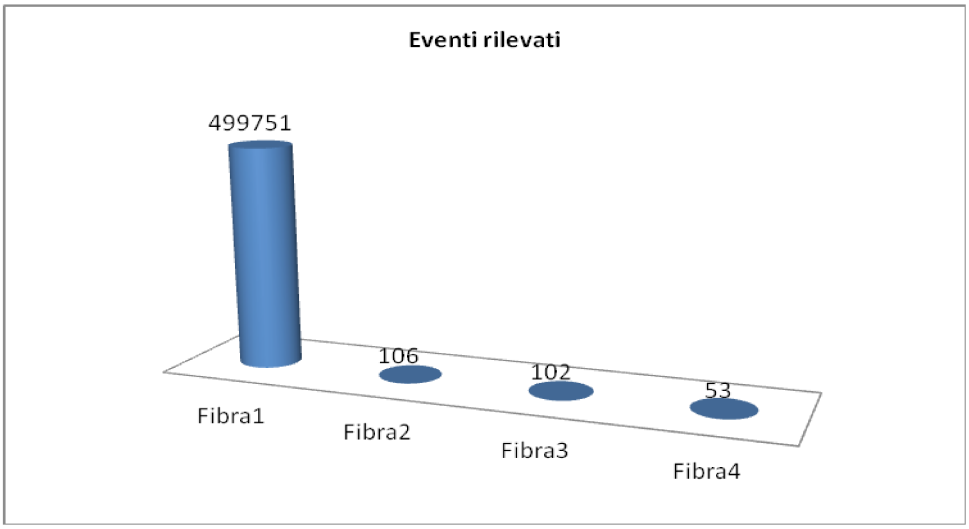
1. una sorgente al centro del fusto di contenimento

```
################ Run 1
                > Source Type della sorgente : Cs137
                > Distribution Type : iso
                > Hits count fibra1: 179
                > Hits count fibra2: 181
                > Hits count fibra3: 191
                > Hits count fibra4: 167
####################################
```



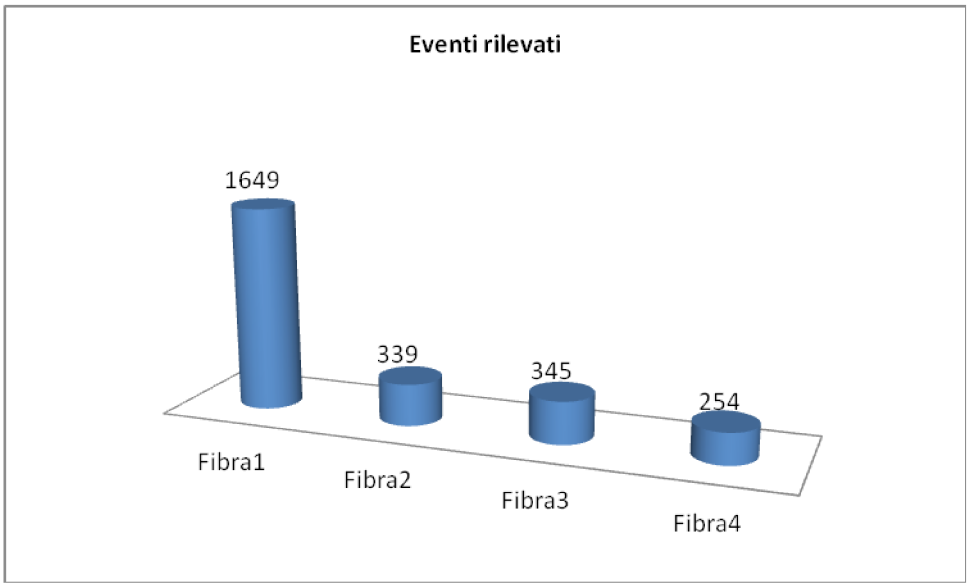

*Eventi contati dal rivelatore in presenza di una sorgente puntiforme posizionata al centro.*

2. una sorgente posta ad 1 cm dalla fibra.



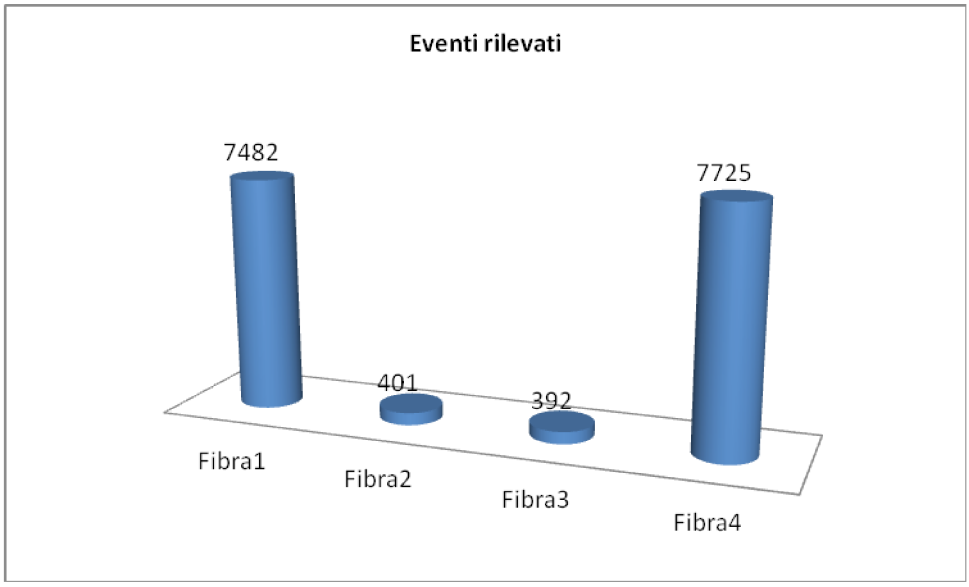

*Eventi contati dal rivelatore in presenza di una sorgente puntiforme posta ad 1 cm dalla Fibra1*

3. due sorgenti poste rispettivamente in posizione centrale e ad 11 cm dalla Fibra1




*Eventi contati dal rivelatore in presenza di due sorgenti puntiformi.*

4. tre sorgenti così disposte: una in posizione centrale, una a 1 cm dalla Fibra1 e l'altra a 1 cm dalla Fibra4.




*Eventi contati dal rivelatore in presenza di tre sorgenti puntiformi*

### 6.2.2 RISULTATI DELLA SIMULAZIONE CON SORGENTE ESTESA

La sorgente estesa è invece un approccio piuttosto innovativo per la generazione di eventi, essa è stata realizzata con l'istanza *G4GeneralParticleSource* sempre della classe *G4VPrimaryGenerator*.

Il codice sottostante mostra l'implementazione della sorgente estesa

```
gun = new G4GeneralParticleSource();
particle = G4ParticleTable::GetParticleTable()->FindParticle("gamma");
```

Come già detto, questa tipologia realizzata per l'emissione di eventi è caratterizzata da una distribuzione random degli stessi, ma, permette, la definizione geometrica della sorgente, in particolare è possibile realizzare sorgenti volumetriche o planari mediante il comando sottostante:

```
/gps/pos/type Volume /Planar.....
/gps/pos/shape Cylinder/Square/ Circle.....
```

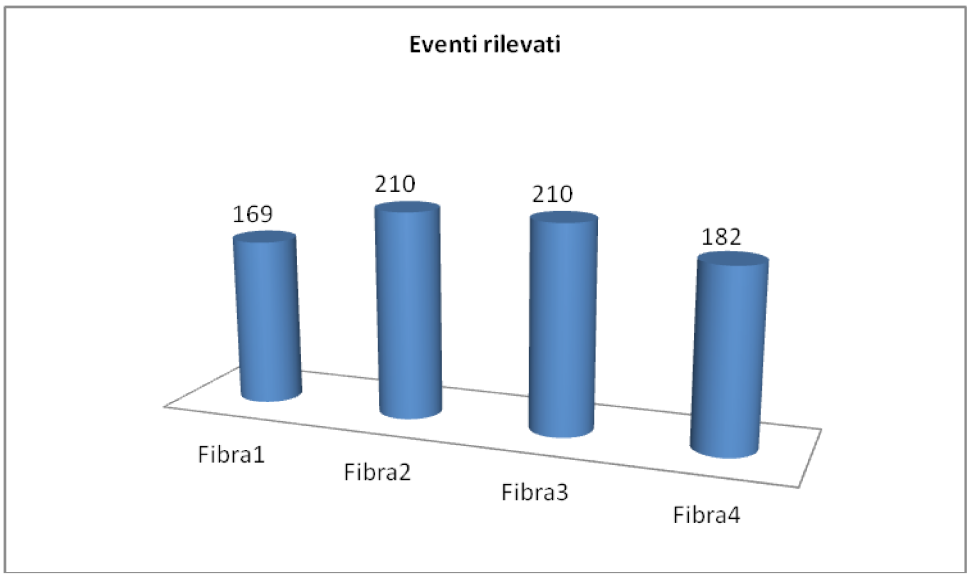
Tuttavia, per la simulazione è stata utilizzata una sorgente cilindrica definita nel *file macro* mostrato sotto:

```
/gps/particle gamma
/gps/pos/type Volume
/gps/pos/shape Cylinder
/gps/pos/centre 0. 0. 0. cm
/gps/pos/radius 0.5 m
/gps/pos/halfx 0.4 m
/gps/pos/halfy 0.4 m
/gps/pos/halfz 0.4 m
```

*Fig. 39: Macro file per la generazione di una sorgente estesa.*

Anche qui le particelle generate saranno tutte rivelate dal detector a meno di un 6% dovuto alle particelle che colpiscono il cladding. Il risultato del conteggio è mostrato in figura.

```
################ Run 4
                > Source Type della sorgente : Cs137
                > Distribution Type : iso
                > Hits count fibra1: 169
                > Hits count fibra2: 210
                > Hits count fibra3: 210
                > Hits count fibra4: 182
###################################
```




*Eventi contati dal rivelatore in presenza di una sorgente estesa.*

I risultati ottenuti dimostrano l'attendibilità del prototipo in accordo con i risultati analitici calcolati.

In virtù della miglior statistica gli spettri di energia ottenuti nel caso di distribuzione puntiforme isotropica risultano più significativi di quelli del caso precedente. Di seguito sono riportati a titolo di esempio quelli relativi alle simulazioni in cui la sorgente è posta al centro del sistema di rivelazione.


*Spettro di energia ottenuto nel caso di sorgente puntiforme al centro del sistema di rivelazione.*

## CONCLUSIONI

Con questo lavoro svolto presso i Laboratori Nazionali del Sud dell'INFN si è cercato di sviluppare un nuovo dispositivo per il monitoraggio delle scorie radioattive contenute nei fusti di un sito di stoccaggio.
Inoltre sono stato effettuati test simulativi sul modello di sensore, con una geometria estesa innovativa, che ha dato risultati molto interessanti, i quali potrebbero aprire nuove strade per sviluppi futuri.
Alla luce dei risultati ottenuti si può concludere che l'esito delle simulazioni è in accordo con quanto ci si aspetterebbe dal comportamento reale dei materiali. Pertanto, l'attendibilità dei risultati raggiunti con lo strumento utilizzato, conducono al prossimo passo, che potrebbe essere quello di simulare l'intero sistema di monitoraggio simulando un qualsiasi numero di fibre attorno ad uno o più fusti disposte in varie geometrie. Il dispositivo si presta bene a tale scopo, grazie soprattutto alle sue caratteristiche, quali:
*costi ridotti;*
*possibilità di avere informazioni tridimensionali* sulla radiazione emessa dalle scorie contenute nei fusti; ciò è reso possibile grazie al fatto che la fibra oltre a presentare una buona flessibilità, ha una lunghezza di attenuazione (1.2 m) adatta al trasporto della luce di scintillazione, prodotta in un punto della fibra dalla radiazione ionizzante proveniente dalle scorie, e rivelata dai SiMP posti alle sue estremità.
*modalità di funzionamento come contatori di eventi:* il rivelatore studiato permette di contare il numero di eventi e quindi il rate degli eventi. Questo permette di stabilire una eventuale rottura del fusto, poichè in conseguenza di tale rottura ci sarebbe un aumento del rate di eventi rivelati. Inoltre questo è reso possibile dal fatto che con il rivelatore studiato è possibile misurare elevati rate di conteggio grazie alla rapidità della risposta sia della fibra che del SiPM.
*resistenza alla radiazione:* durante tutto il periodo di misure la fibra ha dato risultati riproducibili e coerenti con le misure sperimentali. Ovviamente un

utilizzo in campo di questo nuovo dispositivo consentirebbe di validarne in modo completo le prestazioni.

Inoltre, disporre di un maggior numero di fibre attorno al fusto, sia orizzontalmente che verticalmente in modo da formare una griglia, potrebbe dare informazioni più precise. Con questo sistema di rivelazione è possibile individuare oltre che il fusto danneggiato, all'interno del sito di stoccaggio, anche il punto di fuoriuscita del materiale radioattivo del fusto stesso riducendo cosi  il tempo di intervento e quindi la probabilità di incidenti. A tale scopo sicuramente saranno utili ulteriori simulazioni dell'apparato di rivelazione descritto, quando saranno disponibili anche dei dati sperimentali.

**BIBLIOGRAFIA**

[1]http://www.zonanucleare.com/dossier_mondo/depositi_smaltimento_rifiuti_nucleari/A_smaltimento_bassa_radioattivita.htm

[2] Finocchiaro, P., DMNR: a new concept for real-time online monitoring of short and medium term radioactive waste.

[3]Hartmann, S., The World as a Process: Simulations in the Natural and Social Sciences, Hegselmann, R., et al; Modelling and Simulation in the Social Sciences from the Philosophy of Science Point of View,

[4]Metropolis, N., The beginning of the Monte Carlo Method, Los Alamos Science Special Issue, (1987)

[5]Wallin, M., Monte Carlo simulation in statistical physics, KTH (Royal Institute of Technology), SE-100 44 Stockholm, Sweden (2005)

[6]Hammersley, J.M., Handscomb, D.C., Monte Carlo methods, Fletcher, (1975), ISBN 0-416-52340-4

[6]A. Fasso et al. FLUKA: Status and Prospective for Hadronic Applications Advanced Monte Carlo for Radiation Physics, Particle Transport Simulation and Applications, Proceedings of the Monte Carlo 2000 Conference, Lisbon, October 23–26, 2000, edited by A. Kling, F. Barao, M. Nakagawa, L. Tavora, and P. Vaz Springer-Verlag, Berlin (2000) pp. 955–960.

[7] Geant4 Users' Documents Physics Reference Manual
http://geant4.web.cern.ch/geant4/G4UsersDocuments/UsersGuides/PhysicsReferenceManual/html/ PhysicsReferenceManual.html

[8] Applicazioni di Geant4
http://geant4infn.wikispaces.com

[9] Rutherford, E., Geiger, H., An electrical method of counting the number of α particles from radioactive substances, Proceedings of the Royal Society (London), Series A, vol. 81, (1908), no. 546

[10]Leo, W. R., Techniques for Nuclear and particle Physics Experiments, 2nd edition, Springer, (1994), ISBN 354057280

[11] Scintillatori (http://oldweb.ct.infn.it/~rivel/Tipi/Scint/scintillatori.html)

[12] Scintillation product – Scintillating Optical Fibers, Saint-Gobain Crystals

[13]Rutherford, E., The Scattering of α and β Particles by Matter and the Structure of the Atom, Philos. Mag., vol 6,
(http://www.lawebdefisica.com/arts/structureatom.pdf)

[14]GEANT-Detector Description and Simulation Tool
[CERN Geneva, Switzerland, Edition – March 1994];

[15]Diffusione Compton e produzione coerente di pioni neutri su nucleo di $^{4}$He con fotoni etichettati in energia e linearmente polarizzati

[16]Particle Gun & GPS
http://hypernews.slac.stanford.edu/HyperNews/geant4/get/eventtrackmanage/655.html

[17]Finocchiaro, P., DMNR: un dimostratore di sistema per il monitoraggio online in tempo reale di depositi di scorie

[18]Geant4 User's Guide for Application Developer
(http://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/ForApplicationDeveloper/html/inde x.html)

[19]Geant4 Software Reference Manual
(http://geant4.cern.ch/bin/SRM/G4GenDoc.csh?flag=1)

[20]Geant4 Tutorial at Texas A&M 11-Jan-2011

[21]ParticleGun - Makoto Asai (SLAC) Geant4 Tutorial Course

[22]General purpose Source Particle Module for Geant4/SPARSET: Technical Note, UoS-GSPM-Tech, Issue 1.1, C Ferguson, February 2000.

[23] General purpose Source
http://reat.space.qinetiq.com/gps/new_gps_sum_files/gps_sum.htm

[24] The RD44 Collaboration, http://www.infocern.ch/asd/geant/rd44.html.

[25]The Geant4 Collaboration,
http://www.infocern.ch/asd/geant4/geant4.html.

[26]http://geant4.web.cern.ch/geant4/.

[27] Italian Geant4 Developers,
http://www.ge.infn.it/geant4/group/collaborators.html

[28]Mazzoldi, P., et al.: Elementi di fisica - elettromagnetismo, 2nd edition, EdiSES, (2010), ISBN 88-7959-306-4

[29] S. Agostinelli et al.: Geant4 - a simulation toolkit, Nuclear Instruments and Methods in Physics Research A 506 (2003)

[30] Classificazione rifiuti nucleari
http://www.eniscuola.net/it/sostenibilit/contenuti/energia/left/nucleare/lo-smaltimento-dei-rifiuti-nucleari/

[31] Contatori Geiger (http://it.wikipedia.org/wiki/Contatore_Geiger)

[32]J. Allison et al.: Geant4 developments and applications, IEEE Transactions on Nuclear Science 53 No. 1 (2006) 270-278

**RINGRAZIAMENTI**

*Dal punto di vista della stesura di questo lavoro devo ringraziare tutto l'INFN che ha collaborato e partecipato con pazienza e disponibilità alla sua realizzazione.*
*Ringrazio il prof. Malgeri per la sua totale disponibilità e cortesia durante l'attività di tirocinio e soprattutto durante la stesura di questa tesi.*
*Ringrazio il Dr. Paolo Finocchiaro che, con le sue eccellenti conoscenze e con la sua estrema semplicità, ha reso questo progetto davvero interessante.*
*Ringrazio l'Ing. Gianfranco Vecchio per avermi sempre supportata e soprattutto sopportata durante tutto il periodo di tirocinio. Ringrazio la "Squadra fortissimi" Eleonora, Laura e Giusina per aver affrontato insieme a me questo arduo percorso, condividendone le gioie ma soprattutto i dolori :-). Grazie anche alle mie meravigliose coinquiline Giusi, Mary e Laura, per i magnifici giorni e le bellissime serate passate sempre insieme, e grazie anche a voi, Ely e Je, che, pur essendo lontane, siete sempre state presenti.*
*Ma il ringraziamento più grande va ai miei genitori senza i quali certamente non sarei la persona che sono e* senza i quali non sarei mai potuta arrivare a questo punto, *perchè mi siete sempre stati vicini nei momenti più difficili della mia vita e perchè mi avete costantemente sostenuto con assoluto orgoglio, soddisfazione e tanti tanti sacrifici.*
*Ed infine, ma non certo ultimo per importanza, grazie a te Fabio, il ragazzo più importante della mia vita, per aver condiviso con me tutti i momenti belli e difficili di questi anni spronandomi sempre a superarli.*