UNIVERSITÁ DEGLI STUDI DI CATANIA

FACOLTÁ DI INGEGNERIA

Corso di Laurea di I livello in Ingegneria Informatica

TESI DI LAUREA

ASPETTI TECNICI E MECCANISMI DI SIMULAZIONE DI FENOMENI FISICI: LA RADIAZIONE GAMMA NELLE FIBRE SCINTILLANTI

Valentina Finocchiaro

616/002020

Relatore:

Prof. Michele Malgeri

Correlatore:

Dott. Paolo Finocchiaro

Anno accademico 2010 - 2011

Sommario

Introduzione

Il presente lavoro di tesi, svolto presso i Laboratori Nazionali del Sud dell'INFN, va inquadrato in un contesto più ampio che riguarda la problematica del monitoraggio delle scorie radioattive che è uno dei temi di maggiore interesse sia scientifico che sociale e che necessita di studi approfonditi non sempre possibili su sistemi reali data la loro pericolosità.

L'obiettivo di questo lavoro di tesi è quello di creare un modello al *computer* che permetta lo studio qualitativo e quantitativo del problema della rivelazione della radiazione gamma, proveniente dai rifiuti radioattivi, mediante dei rivelatori innovativi attualmente allo studio presso LNS e dei quali è necessario effettuare una caratterizzazione completa e realistica. Il modello permette la simulazione del loro comportamento e della loro risposta in termini di spettro di energia e conteggi in determinate configurazioni di interesse.

Nello specifico il lavoro riguarda lo sviluppo di un'applicazione per simulare la rivelazione della radiazione gamma prodotta da sorgenti radioattive puntiformi attraverso una fibra scintillante di polistirene avvalendosi del *toolkit* Geant4 per la simulazione del passaggio di particelle attraverso la materia, che fa uso di algoritmi Monte Carlo. L'applicazione scritta in linguaggio C++, sfrutta le risorse computazionali per simulare le caratteristiche geometriche e fisiche del rivelatore e le interazioni tra particelle e materiali. Per lo sviluppo è stata necessaria una preliminare analisi del *toolkit* Geant4 e delle librerie presenti da cui derivare quelle specifiche per l'applicazione in questione, e dell'interfaccia per la configurazione dell'ambiente di simulazione, che comprende la geometria del rivelatore, i materiali e i processi fisici coinvolti, le particelle di interesse, le sorgenti di radiazione, etc.

Sono state eseguite diverse simulazioni, variando di volta in volta alcuni parametri del simulatore, che hanno permesso di trarre una valutazione della risposta del rivelatore sulla base dei dati ottenuti consistenti nei conteggi delle particelle intercettate e nello spettro dell'energia rilasciata. Le simulazioni sono state condotte distinguendo tra due tipi di distribuzione dei raggi; una prima serie di simulazioni è

stata eseguita impostando una distribuzione isotropica in tutto lo spazio, con lo scopo di valutare il numero di particelle incidenti per ricavare l'efficienza geometrica della fibra, e una seconda impostando il generatore di particelle in modo tale da emettere raggi gamma selettivamente nell'angolo solido del rivelatore allo scopo di ottenere una migliore valutazione dello spettro energetico e dell'efficienza di rilevazione. Infine, la risposta del rivelatore è stata riportata in diversi formati: in particolare per l'efficienza geometrica è stato tracciato l'andamento al variare della distanza della sorgente dal rivelatore e per l'efficienza di rivelazione è stato riportato l'andamento al variare sia della distanza che della soglia in termini di energia.

L'applicazione è stata sviluppata ed eseguita su un processore *Intel Core Duo* 2 Ghz, Sistema operativo *OS Leopard* 10.5 utilizzando la versione 9.3 p02 del *toolkit*, anche se attualmente è disponibile una versione aggiornata. La tesi è organizzata nel modo seguente:

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

Capitolo 2 : sono discussi gli aspetti riguardanti la simulazione in generale, una introduzione 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 : questo capitolo è dedicato alla descrizione dell'architettura e del *design* del *toolkit* Geant4;

Capitolo 4 : sono esposti gli aspetti fisici che riguardano il problema affrontato, in particolare la fisica che riguarda l'interazione dei raggi gamma con la materia, una breve descrizione dei processi fisici coinvolti nella simulazione e qualche cenno agli scintillatori e una loro classificazione;

Capitolo 5 : contiene la parte riguardante l'applicazione sviluppata, la 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 : infine sono trattati gli aspetti riguardanti le verifiche effettuate, i *run* delle simulazioni eseguite e l'analisi dei dati ottenuti.

1 DMNR: un sistema per il monitoraggio real-time delle scorie radioattive

Presso l'INFN-LNS è attualmente in fase di sviluppo un progetto, denominato DMNR (*Detector Mesh for Nuclear Repositories*)[1], per il monitoraggio *real-time* delle scorie radioattive. Tale progetto, che prevede l'impiego di particolari sensori a basso costo basati sull'utilizzo di fibre scintillanti, è stato recentemente testato attraverso un prototipo presso la ex centrale nucleare del Garigliano situata nel comune di Sessa Arunca in provincia di Caserta, riscuotendo risultati soddisfacenti.

Nel seguito si illustrerà brevemente il sistema di stoccaggio dei rifiuti radioattivi accennando alle tipologie di radiazioni coinvolte ed all'attuale sistema di monitoraggio e descrivendo sommariamente l'alternativa proposta dal progetto DMNR.

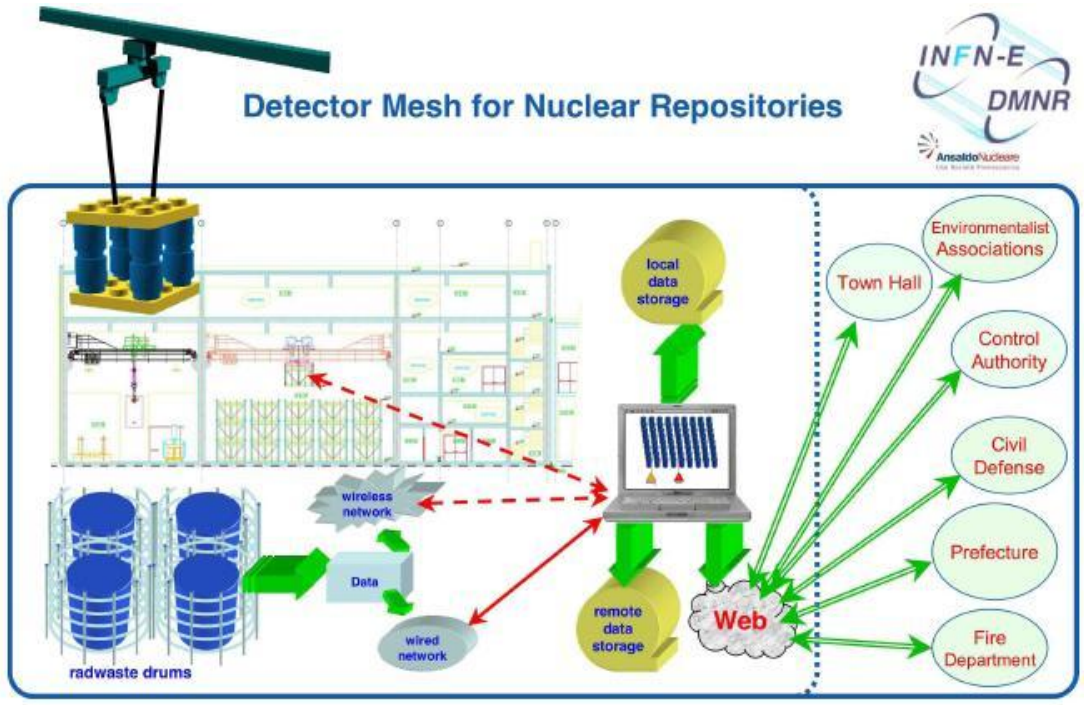


Figura 1. Schema generale del sistema di monitoraggio DMNR.

1.1 I rifiuti radioattivi

Con il termine scorie radioattive si indicano quei rifiuti contenenti materiale radioattivo prodotti come residui di processi nucleari, di processi industriali o di applicazioni mediche. Poiché la radioattività decresce nel tempo, si potrebbe pensare di sigillare e isolare i rifiuti per qualche tempo fino a che il loro livello di radioattività non costituisca più un rischio rilevante. Purtroppo però il tempo necessario affinché questo avvenga varia a seconda della specie radioattiva. In altre parole, l'attesa per ricondurre la radioattività al di sotto della soglia di rischio potrebbe essere misurata in anni per alcune scorie e in decine di migliaia di anni per altre, come quelle altamente radioattive prodotte dai reattori a fissione nucleare. L'attuale procedura di smaltimento di tali rifiuti prevede il loro deposito in fusti (Figura 2), custoditi all'interno del luogo di produzione o trasportati in depositi di stoccaggio, costituiti in materiali pesanti, ad esempio il piombo, che hanno lo scopo di bloccare le radiazioni pericolose prodotte.

Figura 2. Esempio di deposito di fusti contenenti materiale radioattivo.

Le tipologie di radiazioni emesse da un fusto di rifiuti radioattivi sono: alfa, beta, gamma e neutroni dovuti a fissione nucleare[a]. 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.

Figura 3. Schema delle radiazioni e del loro livello di penetrazione, in un comune fusto di rifiuti radioattivi.

1.2 Il monitoraggio

Ogni specie radioattiva presenta una propria "costante di decadimento" e decade seguendo una legge esponenziale. La costante di decadimento rappresenta il tempo necessario per ridurre la "popolazione" di un dato radioisotopo del 37% del valore iniziale. Si deduce che la radioattività decresce con il tempo, ma potrebbero volerci alcuni secondi per alcune specie e milioni di anni per altre. Si pone il problema di monitorare questi rifiuti: l'obiettivo sarebbe la misurazione della radioattività attorno ad ogni singolo fusto,

[a] La fissione nucleare 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.

in modo tale da consentire un intervento tempestivo qualora si verificassero perdite a causa di debolezza strutturale o di incidente. L'attuale sistema di monitoraggio dei rifiuti prevede l'impiego di contatori Geiger-Müller [21], dispositivi per 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 un costo proibitivo. Per evitarlo si ricorre ad un controllo periodico da parte di operatori (o *robot*) o al posizionamento di alcuni sistemi di monitoraggio in punti determinati all'interno dello stesso deposito.

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. Il sistema di monitoraggio, progettato per essere distribuito, robusto, affidabile e basato su componenti poco dispendiosi, propone un rivelatore scintillante economico, equipaggiato con dei fotosensori SiPM (*Silicon Photomultiplier*)[b] 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. 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[c] (*Field Programmable Gate Array*), una interfaccia utente grafica e un sistema di archiviazione dati. E' prevista anche 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

[b] Il Silicon Photomultiplier è un dispositivo sensibile al passaggio dei singoli fotoni costituito da un fotodiodo a valanga costruito su un substrato di silicio.
[c] In elettronica digitale il Field Programmable Gate Array è un circuito integrato programmabile il cui comportamento viene definito via software dall'utente finale.

fusti di rifiuti radioattivi recentemente testato con successo nel deposito della ex centrale del Garigliano in provincia di Caserta.

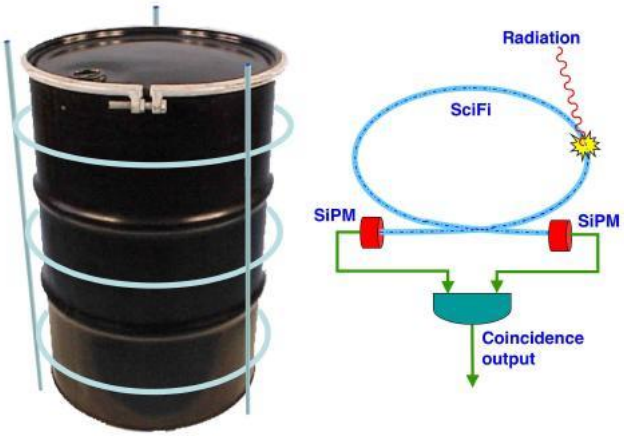


Figura 4. Un possibile arrangiamento della fibra attorno al fusto (a sinistra), e uno schema del principio di funzionamento del sensore (a destra).

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)[d] o il ^{137}Cs (cesio-137)[e]. La sorgente è stata posta a contatto con la fibra in tre diverse posizioni, come mostrato in Figura 5.

[d] 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).
[e] 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.

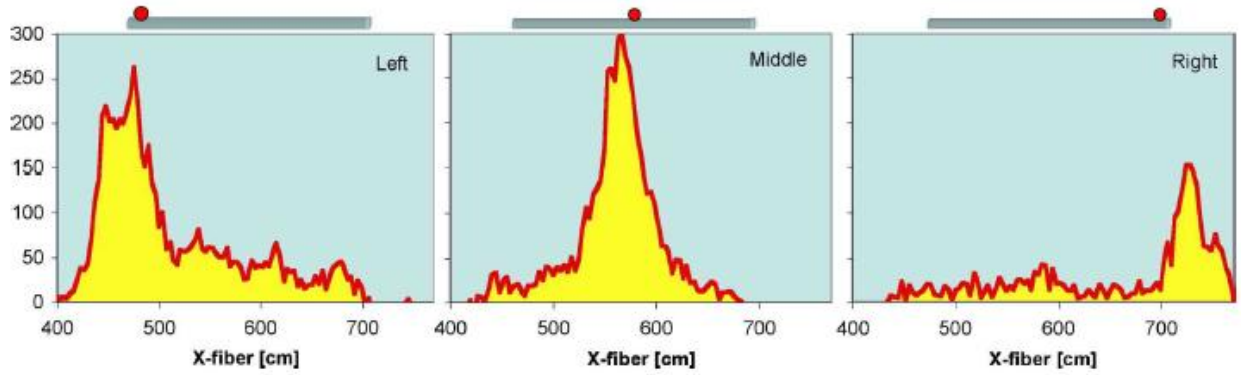


Figura 5. Prova della sensibilità ai raggi gamma della fibra+SiPM. Lungo la fibra è posta una sorgente di ^{60}Co e in corrispondenza è tracciato un istogramma che registra la differenza di tempo tra l'arrivo dei due impulsi di luce ai due sensori SiPM.

I grafici mostrano come effettivamente la fibra sia sensibile alla radiazione gamma, infatti, in corrispondenza della sorgente è visibilmente maggiore il numero di gamma conteggiati. [1]

Per quanto riguarda le simulazioni esse sono state svolte mediante il codice Geant allo scopo di verificare l'efficienza geometrica del sensore. In particolare essa è stata valutata sottoponendo la fibra alla radiazione della sorgente, posta di volta in volta a diverse distanze mediante spostamenti micrometrici. In precedenza è stato usato il codice GEANT3: l'apporto costituito da questo lavoro di tesi a quanto descritto è consistito nell'implementazione del codice per le stesse misure utilizzando il *toolkit* di simulazione Geant4. 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*, GEANT3.21. Gli incoraggianti risultati rendono questo prototipo adatto per applicazioni in molti campi scientifici e tecnologici in cui misure di radioattività possono essere condotte a costo contenuto. [1][2]

2 Simulazione

La simulazione è un potente strumento che consente di studiare l'evoluzione temporale di un sistema reale osservando il comportamento di un sistema artificiale che ne rappresenta un modello computazionale. Simulare quindi significa valutare un processo attraverso un altro, dove per processo si intende una sequenza temporale degli stati di un sistema. 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. Quest'ultimo può essere definito come un insieme di assunzioni fatte su di un sistema allo scopo di rappresentarlo. Si fa distinzione tra modello statico e modello dinamico. Nel primo caso le assunzioni che caratterizzano il modello riguardano un sistema a riposo; nel secondo caso tali assunzioni descrivono l'evoluzione temporale di un dato sistema. Tipicamente un modello dinamico, che rappresenta i fenomeni interessati ad essere indagati tramite simulazione, può essere costituito da equazioni differenziali, nel caso in cui il fenomeno oggetto di studio sia di tipo continuo (nel qual caso si parla appunto di simulazione continua), o attraverso particolari linguaggi per modelli discreti, come il C.A. (*cellular automaton*)[f] per lo studio di fenomeni tempo-discreti (riferendosi in questo caso a simulazione discreta). 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 principalmente 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 (si pensi ad esempio agli

[f] Un automa cellulare è un modello discreto costituito da una griglia regolare di celle, che può avere un numero finito qualsiasi di dimensioni. Ciascuna cella può trovarsi in uno dei possibili stati (in numero finito), ad esempio "On " e "Off".

studi sulla formazione delle galassie). 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"[g]. In ambito scientifico questo strumento costituisce un supporto essenziale per gli esperimenti consentendo un significativo risparmio di risorse in termini di tempo e costi. E' 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.

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 che risultano immediatamente evidenti.[3]

2.1 Il Metodo Monte Carlo

Il metodo Monte Carlo si fonda sulla generazione di numeri casuali per rappresentare il comportamento di un fenomeno caratterizzandolo attraverso la sua statistica. 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 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

[g] Cit. [3], pag. 1

termonucleari fu John Von Neumann, che a quel tempo lavorava al progetto Manhattan 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. Va detto che quindici anni prima Enrico Fermi aveva già 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 approcciabili 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. [5]

L'esempio più tipico di questa applicazione è il metodo "*hit or miss*" che può essere applicato per il calcolo di qualsiasi volume semplicemente inscrivendo il grafico della funzione in un cubo e generando "punti casuali" al suo interno: il rapporto tra il numero di punti che cadono nell'area del grafico della funzione e il numero di punti totali nella scatola convergerà al valore del rapporto tra l'integrale della funzione e il volume della scatola.[4] Di seguito si riporta un esempio del metodo "*hit or miss*" per il calcolo dell'integrale definito di una funzione:

- Sia f(x) definita in [a,b] e limitata, ovvero $\exists\, c \in \mathbb{R} : 0 \leq f(x) \leq c$
- Consideriamo il rettangolo $R = \{(x, y) : a \leq x \leq b,\ 0 \leq y \leq c\}$
- Sia S la porzione di R al di sotto di f(x)

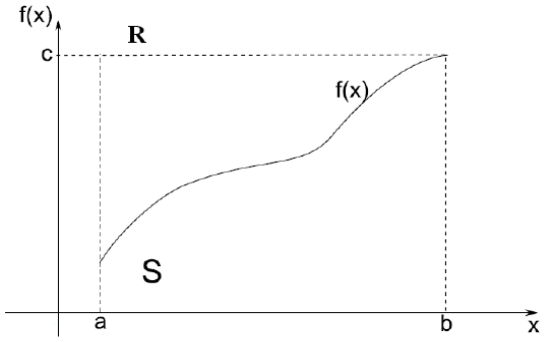


Figura 6

Scelto a caso (x,y) all'interno di R, la probabilità p che cada in S è:

$$p = \frac{area\ di\ S}{area\ di\ R} = \frac{\int_a^b f(x)dx}{c(b-a)} = \frac{I}{c(b-a)}$$

Scelti N campioni casuali in R, una stima di p è N_H/N, quindi una stima I* dell'integrale I è:

$$I^* = c(b-a)\frac{N_H}{N}$$

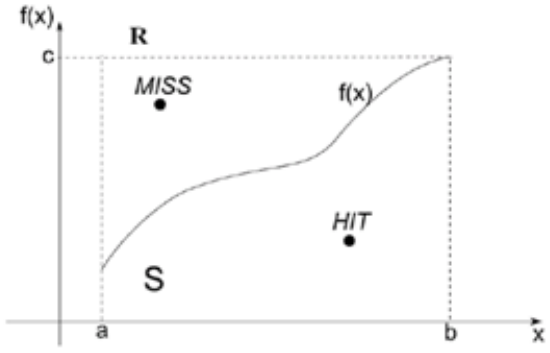


Figura 7

In tal modo si può calcolare ad esempio un'approssimazione del valore π: è sufficiente applicare il metodo per valutare l'integrale sotteso all'arco di circonferenza, come in Figura 8:

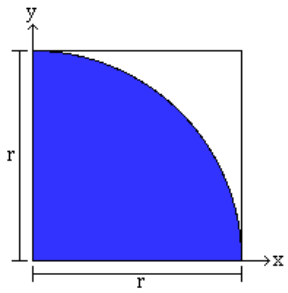


Figura 8

Attraverso il rapporto tra il numero di "*hit*" all'interno dell'area definita dalla semicirconferenza e del numero di "*hit*" totali nell'area del quadrato si può risalire al valore di π con la seguente formula:

$$\frac{N_H}{N} = \frac{area\ della\ semicirconferenza}{area\ del\ quadrato} = \frac{\frac{1}{4}\pi r^2}{r^2} = \frac{1}{4}\pi \rightarrow \pi = 4\frac{N_H}{N}$$

Lo stesso problema può essere risolto mediante un altro approccio Monte Carlo, detto "*Sampling*". [4] Mantenendo le ipotesi fatte in precedenza, vale quanto segue:

- Sia $g_x(x)$ la funzione di distribuzione di probabilità[h] della variabile casuale x.
- $g_x(x_i)$ è la probabilità che sia $x = x_i$

Possiamo scrivere I come:

$$I = \int_a^b \frac{f(x)}{g_x(x)} g_x(x)dx$$

Se $E[f(x)] = \int_a^b f(x)g_x(x)dx$ è il valore medio atteso di f(x) allora vale:

$$I = E\left[\frac{f(x)}{g_x(x)}\right]$$

Se $g_x(x)$ è la funzione di distribuzione uniforme allora possiamo scrivere:

$$I = E\left[\frac{f(x)}{\frac{1}{b-a}}\right] = (b-a)E[f(x)]$$

[h] In statistica e teoria della probabilità, la funzione di ripartizione (o *funzione di distribuzione*) è una funzione di variabile reale che racchiude le informazioni su un fenomeno (un insieme di dati, un evento casuale) riguardanti la sua presenza o la sua distribuzione *prima* o *dopo* un certo punto.

Generiamo N campioni casuali x_i e calcoliamo $f(x_i)$. Una stima di I sarà ottenuta considerando il valore medio $E[f(x)]$ (che nel caso discreto assume la forma seguente):

$$E[f(x)] = \frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

Da cui infine si ottiene la stima:

$$I^* = (b-a)\frac{1}{N}\sum_{i=1}^{N} f(x_i)$$

2.2 Simulazione Monte Carlo

Per sua natura il metodo Monte Carlo è particolarmente adatto allo studio di fenomeni stocastici 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)[i]. Tuttavia le applicazioni del metodo Monte Carlo non si limitano soltanto allo studio dei fenomeni stocastici: è possibile porre la soluzione di un sistema

[i] 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.

deterministico in termini di *pdf* e renderlo adatto ad essere indagato attraverso questi metodi. 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 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. [20]

2.3 Stato dell'arte

Tra i simulatori che utilizzano questo tipo di metodi numerici, in particolare nell'ambito della fisica delle particelle, si possono citare:

- **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.
- **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. 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.

- **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.
- **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 "*Geometry and Tracking*" (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* di

Geant (http://geant4.org) sono disponibili le versioni 9.4 p02 rilasciata il 24 giugno 2011 e la 9.5 Beta rilasciata il 30 Giugno 2011.

3 Geant4 – Geometria e tracciamento

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; per assecondare le esigenze di contenimento della spesa e grazie alla crescente potenza di calcolo offerta dalle nuove tecnologie è stato sviluppato Geant4, un *toolkit object-oriented* per la simulazione del passaggio di particelle attraverso la materia. Il *toolkit* comprende tutti gli aspetti del processo di simulazione - la geometria del sistema, i materiali coinvolti, le particelle di interesse, la generazione di particelle primarie ed eventi, il tracciamento delle particelle attraverso i materiali, etc. - per ognuno dei quali è stato definito un componente *software.* L'interazione delle particelle con la materia è gestita attraverso un ampio set di modelli fisici[j] compresi nel sistema *software*. Al riguardo va detto che i simulatori precedenti presentavano un limite riguardo l'integrazione di nuovi modelli fisici, determinato della forte dipendenza dal codice. Per superare questa difficoltà, che complicava lo sviluppo del sistema *software*, è stato adottato un approccio basato sulla programmazione ad oggetti, che ha consentito lo sviluppo di un *software* molto più flessibile a nuove modifiche. Tale flessibilità si fonda sull'utilizzo di interfacce uniformi e su principi di organizzazione comuni adottati per l'implementazione dei modelli fisici e per la loro integrazione con il *toolkit*. In tal modo l'aggiunta di nuovi modelli diviene una procedura standardizzata che non necessita, se non in casi sporadici, di modifiche al codice esistente. La progettazione di Geant4 è stata sviluppata in un contesto internazionale grazie ad una collaborazione tra Istituti di ricerca e Università. E' stato realizzato grazie alle esperienze di diversi studiosi esperti nella simulazione di rivelatori e processi fisici realizzata con il metodo Monte Carlo.

[j] Nelle scienze notevolmente assiomatizzate, quali la matematica e la fisica, il modello costituisce una possibile interpretazione (o realizzazione) di una teoria, intesa come sistema puramente simbolico e astratto di assiomi e teoremi, in quanto stabilisce un universo di enti per i quali gli assiomi e i teoremi della teoria sono veri o, il che è lo stesso, presenta la stessa struttura logica della teoria; in partic., m. matematico, insieme di enti e relazioni matematiche che consente di descrivere in forma semplificata e controllabile i fenomeni che caratterizzano un sistema in genere complesso. In altre parole un modello costituisce una rappresentazione concettuale del mondo reale o di una sua parte, capace di spiegarne il funzionamento. (cfr. alla voce "Modello" www.treccani.it)

Geant4 rappresenta uno dei progetti più importanti ed ambiziosi nel suo genere per dimensioni, portata del codice ed ampiezza del *team* di lavoro.

3.1 *Design* e architettura

Lo sviluppo del progetto Geant4 si basa su tecniche avanzate di ingegneria del *software*. A causa dell'enorme quantità di codice e della sua complessità, del gran numero di categorie di classi, della vasta area di applicazione, dell'ampiezza e della natura distribuita della collaborazione, Geant4 costituisce un sistema *software* complesso e di grandi dimensioni. La collaborazione, di portata mondiale, ha visto l'impiego di una rigida disciplina ingegneristica in ambito sia progettuale che organizzativo. Nel primo anno di progetto (1995), sono state valutate le metodologie *software* disponibili basate sull'approccio *object-oriented*, scegliendo infine per lo sviluppo di Geant4 il *Booch method*. Tale metodologia, sviluppata da Grady Booch [22], rappresenta nell'ingegneria del *software* un metodo di modellazione a oggetti che comprende una propria notazione grafica per scopi di analisi e progetto di *software.* Sulla base di questo metodo fu in seguito definito lo *Unified Modeling Language* (UML), oggi standard internazionale nella modellazione a oggetti.

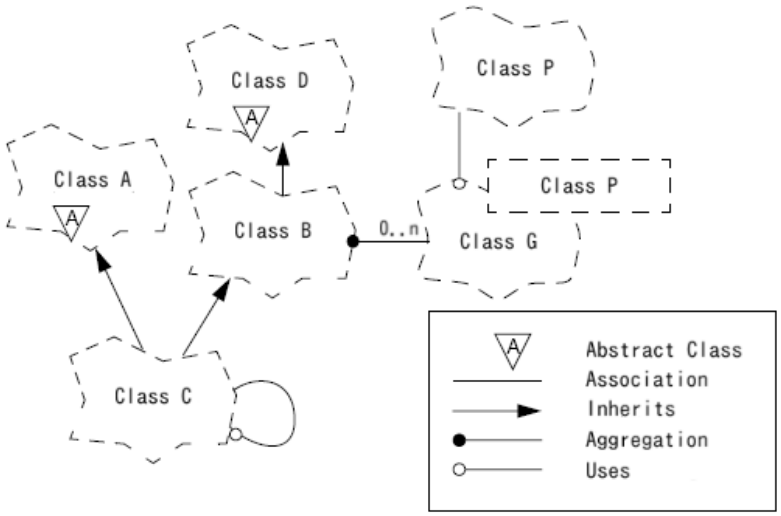


Figura 9. Diagramma delle classi nella notazione grafica Booch

La tecnologia *object-oriented* ha permesso di progettare le classi del *toolkit* in maniera estremamente riutilizzabile e di stabilire una corrispondenza chiara e personalizzabile tra le particelle e i processi.

Come già detto il *toolkit* comprende tutti gli aspetti del processo di simulazione per ognuno dei quali è stato definito un componente *software*. Questi componenti possono essere usati separatamente o in combinazione e devono quindi interfacciarsi in maniera ben definita tra di loro. L'utente ha la possibilità di creare la propria geometria definendo volumi di differenti forme e materiali per costituire il rivelatore, di definire elementi "sensibili" che registrano informazioni utili per simularne la risposta, inoltre, avvalendosi di una varietà di sistemi grafici, può visualizzare la geometria e le tracce.

Il *toolkit* è stato progettato sulla base di alcuni requisiti, tra cui la modularità, la flessibilità e una implementazione della fisica trasparente e facilmente accessibile all'utente. Attraverso un'architettura modulare l'utente può fare uso delle sole componenti di cui ha bisogno nella sua applicazione. La struttura del *toolkit* oltre ad essere modulare è anche gerarchica; i domini chiave della simulazione del passaggio di particelle attraverso la materia sono:

- geometria e materiali;
- interazione di particelle nella materia;
- gestione del tracciamento;
- digitalizzazione e gestione degli *hit*;
- gestione degli eventi e delle tracce;
- visualizzazione;
- interfaccia utente.

Per ciascuno di questi domini è stata definita una categoria di classi con interfaccia uniforme. Questa impostazione realizza il concetto di *toolkit*: l'utente ha la possibilità di assemblare il proprio programma a

tempo di compilazione attingendo dai componenti messi a disposizione dal *software* oppure sviluppando le proprie classi sulla base di quelle predefinite.

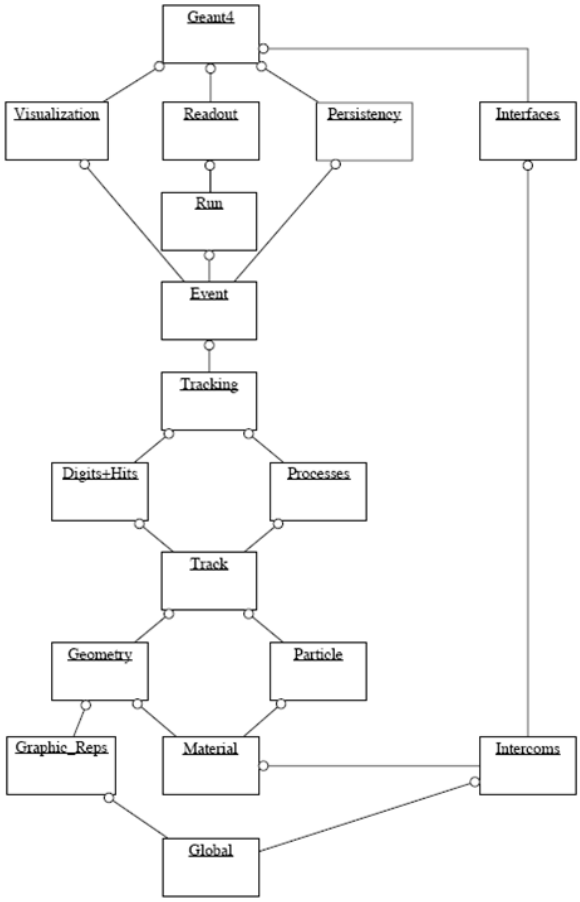


Figura 10. Diagramma delle categorie di Geant4. Il cerchio nelle linee di congiunzione rappresenta una relazione, la categoria adiacente al cerchio utilizza la categoria congiunta.

Il *design*, che si è evoluto nel corso dello sviluppo, attualmente comprende diciassette categorie principali mostrate in Figura 10. Ciascuna categoria si lega alle altre attraverso un flusso unidirezionale di dipendenze; le categorie nella parte inferiore dello schema sono utilizzate da quasi tutte le categorie superiori e costituiscono le fondamenta del *toolkit*. Esse includono la categoria *global, materials*, *particles*, *graphic representations*, *geometry* e *intercoms*. Sopra questi moduli risiedono le categorie necessarie per descrivere il tracciamento delle particelle e i processi fisici a cui sono sottoposte: *track*, *hits e digitization*, *tracking*, *events*, *run*, *readout*. Infine le funzionalità che utilizzano tutte queste categorie e le collegano a

moduli esterni al toolkit attraverso interfacce astratte sono fornite dalle categorie *visualization*, *persistency* e *interfaces*. Di seguito sono descritte brevemente le categorie principali:

- La categoria *track* contiene le classi per le tracce (`G4Track`) e gli step (`G4Step`) usate dalla categoria *processes*, che contiene l'implementazione dei modelli per le interazioni fisiche. Dal momento che le tracce sono rappresentate da oggetti in linguaggio C++, esse possono essere gestite abbastanza semplicemente attraverso contenitori standard: la gestione degli eventi in Geant4 prevede tre *stack* (gestiti con politica *first-in-last-out)*: "*urgent*", "*waiting*" e "*postpone to next event*". In questo modo Geant4 consente di controllare facilmente le priorità usando i due *stack urgent* e *waiting*, a differenza di quanto accadeva nelle versioni precedenti in cui vi era un solo *stack* e ad ogni traccia era assegnata una priorità. Con questa gestione delle tracce l'utente può scegliere in quale *stack* memorizzare la traccia implementando una propria classe derivata da `G4UserStackingAction`.
- La categoria di tracciamento (*traking*) guida semplicemente l'invocazione dei processi fisici registrati per le particelle della simulazione: analizza tutti i processi fisici e le azioni per la data particella e decide quale deve essere richiamato; il tracciamento non dipende dal tipo di particella, né dal processo fisico specifico: tutti i processi fisici connessi alla particella propongono uno *step*, che per una particella a riposo sarà un "tempo" piuttosto che una "lunghezza", e il *tracking* si occuperà di scegliere uno *step* tra quelli proposti dai processi. Tutti i processi fisici sono conformi all'interfaccia di base `G4VProcess`, anche se sono stati sviluppati approcci diversi per la progettazione dettagliata dei sottodomini, in quanto in alcuni casi la complessità dei processi ha richiesto una ulteriore scomposizione (Figura 11). Ad esempio la categoria che riguarda i processi di fisica elettromagnetica è organizzata come un insieme di sottocategorie di classi, in cui ciascuna categoria gestisce un particolare tipo di processi (per es. *standard* gestisce i processi di base per le interazioni di elettroni, positroni, fotoni e adroni; *low energy* fornisce modelli alternativi estesi verso energie più basse rispetto alla categoria standard; *muoni* gestisce le interazioni di muoni, etc.). Il *package* comprende processi di ionizzazione, *bremsstrahlung*, *scattering* multiplo, *Compton*

e *Rayleigh scattering*, effetto fotoelettrico, conversione di coppie, annichilazione, radiazione da sincrotrone e transizione, scintillazione, rifrazione, riflessione, assorbimento ed effetto *Cherenkov*. L'insieme dei modelli fisici alla base dei processi è stato definito adottando una strategia *design pattern*[k] allo scopo di renderli intercambiabili. Infatti questa strategia rende il sistema aperto alle continue evoluzioni che caratterizzano la disciplina della fisica e non richiede modifiche interne in seguito all'introduzione di nuove funzionalità, richiedendo soltanto che un eventuale nuovo algoritmo soddisfi l'interfaccia astratta comune.

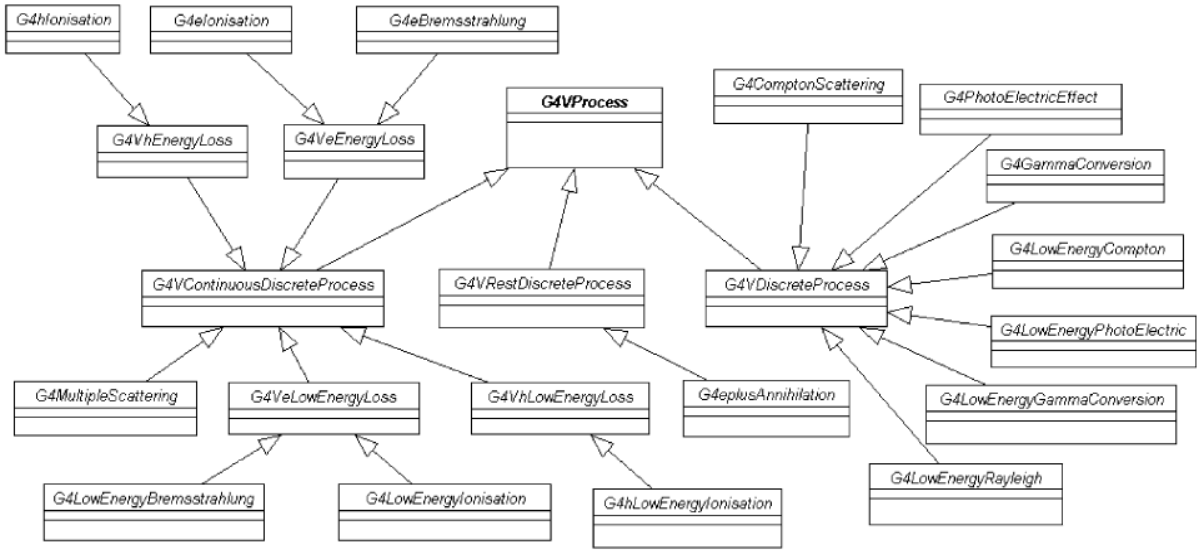


Figura 11 - Diagramma delle classi dei processi elettromagnetici; mostra come sono forniti i processi alternativi, obbedendo alla stessa interfaccia astratta.

- Le categorie *events* e *run* gestiscono rispettivamente gli eventi e una collezione di questi: un evento costituisce l'unità principale della simulazione ed è rappresentato attraverso una istanza della classe `G4Event` che evita di mantenere informazioni transitorie non significative dopo il

[k] Nell'ingegneria del *software*, un *design pattern* (schema di progettazione) può essere definito "una soluzione progettuale generale a un problema ricorrente". Esso non è una libreria o un componente di software riusabile, quanto piuttosto una descrizione o un modello da applicare per risolvere un problema che può presentarsi in diverse situazioni durante la progettazione e lo sviluppo del *software*.

completamento dell'elaborazione dell'evento in questione. L'utente può memorizzare l'oggetto di questa classe per elaborare le informazioni a valle della catena del programma.

- La categoria *geometry* offre la possibilità di descrivere una struttura geometrica e propagarvi particelle attraverso. E' possibile costruire la geometria del rivelatore utilizzando i concetti di volume logico e volume fisico. Un volume logico rappresenta un elemento del rivelatore con determinati attributi che consistono nel materiale di cui è composto, nel comportamento associato alle zone sensibili, nella forma, e in altre informazioni indipendenti dalla sua posizione fisica nel rivelatore. Un volume fisico rappresenta il posizionamento spaziale del volume logico rispetto ad un *mother volume* che lo contiene. In questo modo si costituisce una struttura ad albero gerarchico dove ogni volume contiene volumi più piccoli, non sovrapponibili. In Geant4 il concetto di volume logico è stato migliorato rispetto alle versioni precedenti racchiudendo le caratteristiche relative alla forma del volume in una entità separata, chiamata "*solido*", rappresentata, nel caso di forme semplici, attraverso oggetti delle classi `G4Box`, `G4Tubs`, `G4Sphere`, etc.
- Le due categorie *particles* e *materials* implementano le strutture necessarie per descrivere le proprietà fisiche di particelle e materiali per la simulazione di interazione tra particelle e materia. Le particelle sono definite sulla base della classe `G4ParticleDefinition`, la quale ne descrive proprietà come la massa, la carica, etc. e consente di associare a ciascuna particella la lista dei processi ai quali è sensibile. Tutte le classi concrete che definiscono le particelle come `G4Electron`, `G4Positron`, `G4Gamma`, etc., sono istanziate come "*singleton*"[l] per assicurare che tutti i processi fisici si riferiscano alle stesse proprietà per ciascun tipo di particella. La categoria dei materiali riflette quello che esiste in natura: i materiali sono composti da un insieme di elementi che a loro volta sono composti da isotopi o da una loro combinazione. Le proprietà fisiche dei materiali sono descritte da quantità note, come la densità, o derivate dalla composizione di elementi.

[l] In ingegneria del software, per implementare il concetto matematico di singleton, si utilizza un *design pattern* denominato anch'esso "*Singleton*" (singoletto) per limitare la creazione di istanze di una classe ad un solo oggetto. Ciò è utile quando per coordinare le azioni del sistema è necessario un solo oggetto di una data classe.

- La categoria *Hits+Digitization* permette la gestione della parte sensibile del rivelatore attraverso la generazione di *hit* e *digit* che costituiscono la risposta del *sensitive detector* o parte sensibile del rivelatore, alle particelle che lo attraversano. L'utente deve fornire la propria implementazione della risposta del rivelatore definendo gli oggetti *hit* derivando la classe astratta `G4VHit`. Un *hit* è un'istantanea di una interazione fisica o un accumulo di interazioni di una traccia, o di più tracce, nel componente sensibile del rivelatore. Un *digit* rappresenta l'*output* del rivelatore: è creato da uno o più *hit* e/o altri *digit*. Ogni volume logico può avere un puntatore ad un rivelatore sensibile, il quale è rappresentato da un oggetto implementato dall'utente attraverso una classe derivata dalla classe base astratta `G4VSensitiveDetector`. Il *sensitive detector* crea *hit* usando le informazioni date dallo *step* corrente. Al momento del tracciamento, quando lo *step* si trova all'interno del volume che contiene un puntatore ad un sensitive detector, questo viene invocato con le informazioni sullo *step* corrente.

3.1.1 *User interface*

La categoria *intercoms* fornisce sia il mezzo per interagire con Geant4 attraverso l'interfaccia utente sia un modo per far comunicare tra loro i vari moduli. L'interfaccia 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.

3.1.2 *User Actions*

Geant4 fornisce un'interfaccia astratta per 8 "*user classes*". L'implementazione, l'istanziazione e la registrazione di queste classi sono obbligatorie in tre casi, opzionali in altri cinque. Questo permette all'utente di personalizzare Geant4 per la specifica situazione. Le tre "*mandatory user classes*" che l'utente dovrà derivare per definire gli elementi necessari alla propria applicazione sono:

- `G4VUserDetectorConstruction`: per definire i materiali e la geometria del *detector* e altre proprietà come le zone sensibili del *detector* e gli attributi di visualizzazione.

- `G4VUserPhysicsList`: per definire tutte le particelle, i processi fisici e i parametri di "*cut-off*"[m].

[m] Il *cut-off* è un valore di soglia, definito per le particelle che intervengono nella simulazione, tipicamente rappresentato da una distanza o come un *range* e internamente convertito in energia; rappresenta quella soglia al di sotto della quale non vengono generate particelle secondarie. Deve essere impostato nella fase di inizializzazione all'interno del metodo `SetCuts()` della classe `G4VUserPhysicsList`. [15]

- `G4VUserPrimaryGeneratorAction`: per generare i vertici primari e le particelle.

Per queste tre *user classes*, Geant4 non fornisce un comportamento di *default*; esistono delle definizioni astratte che l'utente deve derivare, implementando da queste le proprie classi concrete. Per esempio, dal momento che è impossibile fornire un insieme di processi che sicuramente saranno applicabili in ogni situazione, Geant4 non definisce dei processi fisici di *default*. Anche il trasporto di particelle (*transportation process*) deve essere registrato dall'utente, altrimenti Geant4 non trasporterà nessuna particella.

Le "*optional user classes*" permettono all'utente di modificare il comportamento di default di Geant4 e sono:

- `G4UserRunAction`: per le azioni da eseguire all'inizio e alla fine di ogni *run*;
- `G4UserEventAction`: per le azioni da eseguire all'inizio e alla fine di ogni evento;
- `G4UserStackingAction`: per gestire l'accesso allo *stack* che contiene le tracce;
- `G4UserTrackingAction`: per azioni da eseguire durante la creazione o al completamento di ogni traccia;
- `G4UserSteppingAction`: per gestire e personalizzare il comportamento ad ogni *step*.

Per sviluppare la propria applicazione l'utente deve fornire la propria implementazione delle *mandatory user classes* derivandole da quelle del *toolkit* ed eventualmente implementare le *optional user classes*. Queste classi dovranno poi essere registrate nel *main* attraverso l'oggetto *run manager* della classe `G4RunManager` che si occuperà dell'inizializzazione.

4 Aspetti fisici

In questo capitolo sono descritti brevemente i processi di interazione tra le particelle utilizzati nella simulazione, gli scintillatori e una loro classificazione.

4.1 Interazione di radiazione gamma con la materia

In fisica nucleare i raggi gamma, indicati con la lettera greca minuscola 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.

Il decadimento gamma è uno dei tre tipi di decadimento radioattivo che esistono in natura (alfa, beta, gamma) che consiste nell'emissione di una particella γ da parte di un nucleo instabile. La particella emessa, ovvero un fotone ad alta energia (o onda elettromagnetica ad alta frequenza), non è dotata né di massa né di carica ed è quindi molto penetrante, al punto da riuscire ad attraversare anche strati di metallo pesante come il piombo.

La rivelazione dei gamma è legata al processo di perdite di energia del fotone quando questo attraversa la materia che costituisce il rivelatore. In termini di ionizzazione, la radiazione gamma interagisce con la materia in tre modi principali: l'effetto fotoelettrico, lo *Scattering Compton* e la produzione di coppie elettrone-positrone. [8]

- Effetto fotoelettrico: avviene quando un fotone gamma interagisce con un elettrone orbitante attorno ad un atomo e gli trasferisce tutta la sua energia, col risultato di espellere l'elettrone dall'atomo. L'energia cinetica del "fotoelettrone" risultante è uguale all'energia del fotone gamma incidente meno l'energia di legame dell'elettrone. L'effetto fotoelettrico rappresenta il meccanismo

principale di interazione tra le particelle gamma al di sotto dei 50 keV, ed è invece meno importante ad energie più alte. [8]

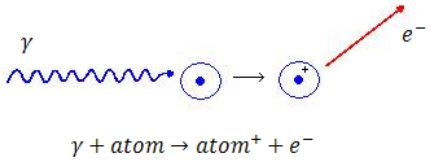


Figura 12 – Effetto fotoelettrico

- *Scattering Compton*: un fotone gamma incidente espelle un elettrone da un atomo, in modo simile al caso precedente, ma l'energia addizionale del fotone viene convertita in un nuovo fotone gamma, meno energetico, con una direzione diversa dal fotone originale. La probabilità dello *scattering Compton* diminuisce con l'aumentare dell'energia del fotone. Si pensa che questo sia il meccanismo principale per l'assorbimento dei raggi gamma nell'intervallo di energie "medie", tra 100 keV e 10 MeV (milioni di elettronvolt), dove va a ricadere la maggior parte della radiazione gamma prodotta da un'esplosione nucleare. Il meccanismo è relativamente indipendente dal numero atomico del materiale assorbente. [8]

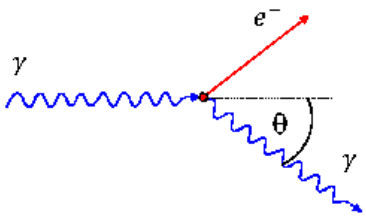


Figura 13 – Effetto Compton

- Produzione di coppie: interagendo con la forza *coulombiana* del nucleo, l'energia del fotone incidente è convertita nella massa di una coppia elettrone/positrone (un positrone è un elettrone carico positivamente). L'energia eccedente la massa a riposo delle due particelle (1.02 MeV) appare come energia cinetica della coppia e del nucleo. L'elettrone della coppia, in genere chiamato elettrone secondario, è molto ionizzante. Il positrone avrà vita breve: si ricombina entro

10^{-8} secondi con un elettrone libero. L'intera massa delle due particelle viene quindi convertita in due fotoni gamma con un'energia di 0.51 MeV ciascuno. [8]

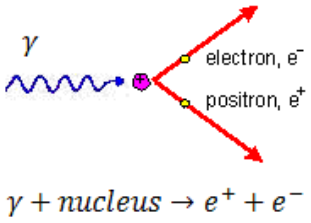


Figura 14 – Produzione di coppia

Per quanto riguarda le particelle secondarie prodotte in seguito a queste interazioni, anche esse sono soggette ad ulteriori interazioni con la materia. In particolare i processi che intervengono e che sono stati considerati per le simulazioni sono:

- Ionizzazione: processo che consiste in una scissione, dovuta alla collisione tra particelle, in seguito alla quale un atomo perde uno o più elettroni che possono restare liberi o essere catturati da altri atomi; ne consegue, in generale, la creazione di una coppia di particelle, una positiva (lo ione che si forma dall'atomo privato di un elettrone) e una negativa (l'elettrone in questione o lo ione in cui si trasforma un atomo che lo cattura).[n] Il distacco di elettroni da un atomo, in cui sostanzialmente la ionizzazione consiste, può avvenire se agli elettroni in questione viene fornita energia (energia di ionizzazione). Naturalmente, più gli elettroni sono vicini al nucleo (come nei non metalli) tanto maggiore è la quantità di energia che deve essere somministrata per allontanare gli elettroni dall'atomo. Tra le molte cause di ionizzazione figurano l'interazione con radiazioni elettromagnetiche (luminose, ultraviolette, X, γ), l'urto con particelle cariche (elettroni, ioni, nuclei atomici), l'agitazione termica. Tale fenomeno quando si manifesta a causa di fotoni, prende il nome di fotoionizzazione[n]. Se un fotone di sufficiente energia, interagisce con gli atomi di un mezzo

[n] cfr. alla voce "Ionizzazione" www.treccani.it

materiale, vengono estratti da questi degli elettroni periferici, dando quindi luogo alla creazione di elettroni liberi.

- *Scattering di Rayleigh*: consiste nella diffusione di un'onda luminosa provocata da particelle con dimensioni molto inferiori rispetto alla lunghezza d'onda della luce, che avviene quando questa attraversa un mezzo sostanzialmente trasparente, soprattutto gas e liquidi. Lo *scattering di Rayleigh* della luce solare da parte delle molecole dell'aria è il motivo principale per cui vediamo il cielo di colore azzurro. La radiazione diffusa è detta anche radiazione *Rayleigh*. [10]

- *Bremsstrahlung*: tale processo si riferisce alla radiazione elettromagnetica prodotta in seguito alla decelerazione a cui sono soggette le particelle cariche quando vengono deflesse a causa dell'interazione con altre particelle cariche (tipicamente un elettrone e un nucleo). Tale radiazione prende il nome di "radiazione *bremsstrahlung"* o "radiazione di frenamento". Dal momento che l'intensità delle onde emesse è inversamente proporzionale al cubo della massa, il *bremsstrahlung* degli elettroni è il più comune in quanto essi sono molto più leggeri dei protoni (massa a riposo circa duemila volte inferiore), mentre per le particelle cariche pesanti, quali i protoni, il fenomeno è trascurabile. [11]

- *Coulomb Scattering:* meglio conosciuto come *Scattering di Rutherford* si riferisce al fenomeno osservato dallo scienziato Ernest Rutherford quando irradiò una lamina d'oro con un fascio di particelle alfa per determinare la struttura dell'atomo, esperimento in seguito al quale si giunse al cosiddetto modello Rutherford dell'atomo; effettivamente i risultati ottenuti evidenziarono una deflessione significativa delle particelle irradiate (1 particella alfa su 8000 subiva una deflessione con un angolo maggiore di 90°) che confermò la presenza di una forte carica concentrata nel nucleo dell'atomo. Questo tipo di processo fisico viene indicato anche con il nome di *Scattering Coulombiano* poiché l'interazione in gioco nell'urto è la forza di *Coulomb*. [12]

- Annichilazione: si parla di annichilazione quando una particella e la corrispondente antiparticella interagiscono dando luogo a particelle più leggere: la differenza tra la massa delle particelle interagenti e quella delle particelle prodotte si ritrova in energia, cinetica o di altro tipo. Nell'annichilazione elettrone-positrone può succedere che le due particelle interagenti si convertano integralmente in energia elettromagnetica (fotoni).[o]

4.2 Scintillatori

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. [9]

4.2.1 Classificazione degli scintillatori

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: scintillatori organici e scintillatori inorganici. ([9] p. 159)

[o] cfr. alla voce "Annichilazione" www.treccani.it

- 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).

- 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 β e γ. 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 ^{6}Li. 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. [14]

4.2.2 La fibra scintillante

Le fibre scintillanti sono sostanzialmente dei fili di scintillatore plastico di diametro variabile fra un centinaio di *micron* e qualche millimetro. Quelle comunemente usate sono costituite principalmente da polistirene, una molecola complessa contenente un anello di benzene (Figura 15). Il corpo di polistirene è contenuto in una o due guaine (*cladding*) con uno spessore totale di qualche *micron* e con indice di rifrazione inferiore a quello del *core* di polistirene: 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.

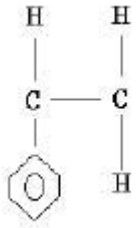


Figura 15 – Molecola di polistirene, l'agglomerato esagonale rappresenta la molecola di benzene contenuta.

Lo scintillatore impiegato nel prototipo DMNR che è stato simulato con Geant4 è proprio una fibra di polistirene (Figura 16);

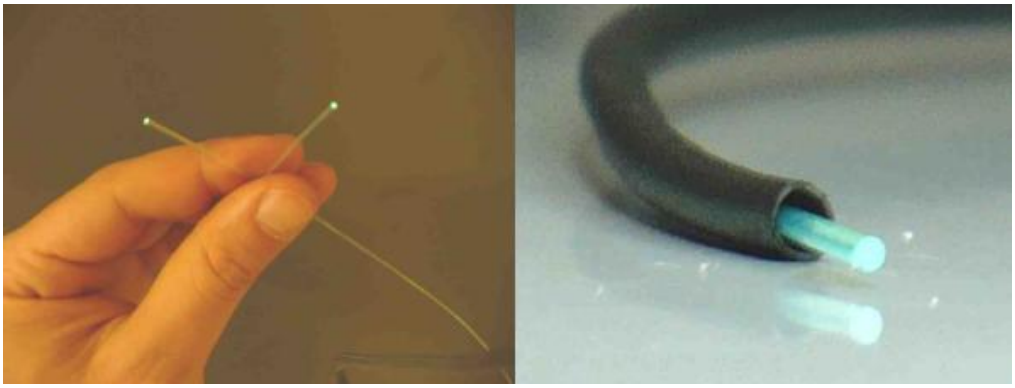


Figura 16 – Una fibra scintillante. A destra è mostrata all'interno di una guaina nera che serve come protezione meccanica e scudo di luce.

uno dei motivi per cui si impiega questo tipo di materiale sta nel vantaggio di poterlo adattare facilmente alla geometria del sistema, consentendo inoltre rapide modifiche; per esempio, si può facilmente tagliare della lunghezza desiderata e disporre intorno alla sorgente in quantità e nella forma che si vuole in quanto è estremamente flessibile. Le principali proprietà di questa fibra sono elencate in Figura 17. ([1], [13])

Fiber type	BCF-20
Core material	Polystyrene
Cladding material	Acrylic
No. of H atoms per cc (core)	4.82×10^{22}
No. of C atoms per cc (core)	4.85×10^{22}
No. of electrons per cc (core)	3.4×10^{23}
Core refractive index	1.60
Cladding refractive index	1.49
Density	1.05
Numerical aperture	0.58
Cladding thickness, round fibers	3% of fiber diameter
Cladding thickness, square fibers	4% of fiber size
Trapping efficiency, round fibers	3.44% minimum, at each end
Trapping efficiency, square fibers	4.4% minimum, at each end
Emission color	Green
Wavelength of emission peak (nm)	492
Decay time (ns)	2.7
1/e attenuation length (m)	>3.5
Light yield (photons/MeV)	~8000
Operating temperature	-20°C to +50°C
Vacuum compatible	Yes

Figura 17 - Proprietà della fibra scintillante attualmente impiegata nel progetto DMNR.

5 Applicazione sviluppata

In questo capitolo sono riportati i dettagli del codice sviluppato per simulare la risposta di una fibra scintillante di polistirene sottoposta ad una sorgente radioattiva di ^{60}Co. Sarà posta l'attenzione sull'architettura generale dell'applicazione e sulle classi di maggiore interesse definite per la costruzione della geometria del rivelatore, per la registrazione delle particelle e dei processi fisici che sono stati presi in considerazione e per il *setup* delle simulazioni.

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. 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:

- `DetectorConstruction`: in cui è definita la geometria del *detector*, i materiali utilizzati e le parti sensibili del volume.
- `PhysicsList`: in cui sono definiti i processi fisici e le particelle coinvolti nelle simulazioni.
- `PrimaryGeneratorAction`: specifica come deve essere generato un evento primario, cioè il modo in cui le particelle primarie vengono generate, specificando energia, posizione e distribuzione.

Nel seguito saranno illustrati i dettagli che riguardano queste tre classi obbligatorie e le *optional user classes* 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.

Prima di cominciare a parlare delle classi implementate diamo una descrizione della struttura generale dell'applicazione attraverso il metodo `main`, nel quale devono essere definiti alcuni oggetti *manager* fondamentali per consentire l'assemblaggio degli "strumenti" occorrenti al simulatore e definiti attraverso le classi sopra elencate.

5.1 Il main

Geant4 non fornisce un metodo `main()` di *default*: l'utente dovrà implementarlo sulla base delle proprie esigenze. Ci sono tuttavia dei passi da seguire per costruire l'applicazione. In particolare 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,*
- le particelle e i processi fisici da simulare,
- il *setup* dell'evento primario.

Come mostrato nel Codice 1 i primi due punti vengono inizializzati attraverso il metodo `SetUserInitialization()` della classe `G4RunManager`, che riceve come parametro il puntatore ai rispettivi oggetti delle *mandatory user classes*.

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

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

G4VUserPhysicsList* physics = new PhysicsList();
```

```
    runManager->SetUserInitialization(physics);
```

Codice 1 – 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()`, che riceve il puntatore all'istanza della classe (Codice 2). 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*.

```
G4VUserPrimaryGeneratorAction* gen_action = new PrimaryGeneratorAction();
runManager->SetUserAction(gen_action);
```

Codice 2 – Creazione e inizializzazione dell'oggetto della *user action class* PrimaryGeneratorAction

Dopo aver passato al `runManager` tutti i puntatori agli oggetti delle *initialization classes* si chiama il metodo `Initialize()` che procederà a sua volta all'invocazione dei metodi necessari per la costruzione del *detector*, dei processi fisici, etc.

`G4UIManager` è la classe *manager* della categoria *intercoms*, 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++ (nel qual caso potrebbe essere necessario ricompilare, qualora si volessero cambiare i parametri, ad esempio il numero di eventi da generare), oppure in modalità *batch* leggendo da opportuni file di comandi, detti "*macro file*". Tuttavia, per evitare una quantità eccessiva di codice e per rendere l'applicazione interattiva, la categoria *intercoms* fornisce la classe astratta `G4UISession` che consente di "catturare" comandi interattivi attraverso diverse interfacce utente (testuali e grafiche) rese disponibili dalla categoria *interfaces* tra le quali l'utente può scegliere.

L'implementazione del metodo `main` per l'applicazione, come mostra il Codice 3, prevede la possibilità di scegliere tra due interfacce testuali rappresentate dalle classi `G4UIterminal` e `G4UItcsh` (la prima 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;
}
```

Codice 3 – 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 4.

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

Codice 4 – Creazione e inizializzazione del Visualization Manager

L'utilizzo della variabile d'ambiente `G4VIS_USE` consente di costruire facilmente un eseguibile privo di visualizzazione, qualora richiesto, senza cambiare il codice sorgente. 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*.

Un esempio di *macro* utilizzata per la visualizzazione del *detector* e delle traiettorie dei raggi gamma è riportato nel codice seguente:

```
# Create a scene handler for a specific graphics system
# (Edit the next line(s) to choose another graphics system)
```

```
#
#/vis/open OGLSX 600x600-0+0
#/vis/open OGLIQt 600x600-0+0
#
#/vis/open OGLIXm
#/vis/open OGLSXm
#/vis/open HepRepXML
#
#/vis/open DAWNFILE
#
# draw scene
#

# VIEWER-0
/vis/open OGLIX 600x600-0+0
/vis/drawVolume
/vis/viewer/set/style surface
/vis/viewer/set/viewpointThetaPhi 0 0 deg
/vis/viewer/zoomTo 1000
/vis/scene/add/axes

# VIEWER-1
#/vis/open OGLIX 800x800-0+0
#/vis/drawVolume
#/vis/viewer/set/style surface
#/vis/viewer/set/viewpointThetaPhi 0 0 deg
#/vis/viewer/zoomTo 100
#/vis/scene/add/axes 0 0 0

# VIEWER-2
/vis/open OGLIX 600x600-0+100
/vis/drawVolume
/vis/viewer/set/style surface
/vis/viewer/set/viewpointThetaPhi 90 0 deg
/vis/viewer/zoomTo 100
/vis/scene/add/axes 0 0 0

#/vis/open HepRepFile
#/vis/drawVolume
#/vis/viewer/flush

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

# (if too many tracks cause core dump => /tracking/storeTrajectory 0)
# for drawing the hits, uncomment next line
/vis/scene/add/hits

# (if you prefer refreshing each event, comment out next line)
# /vis/scene/endOfEventAction accumulate N
/vis/scene/endOfEventAction accumulate 1000
```

Codice 5 – Esempio di *macro file* utilizzato per la visualizzazione del *detector*

Il codice contenuto nel *macro file* `vis.mac` genera le seguenti visualizzazioni (Figura 18):

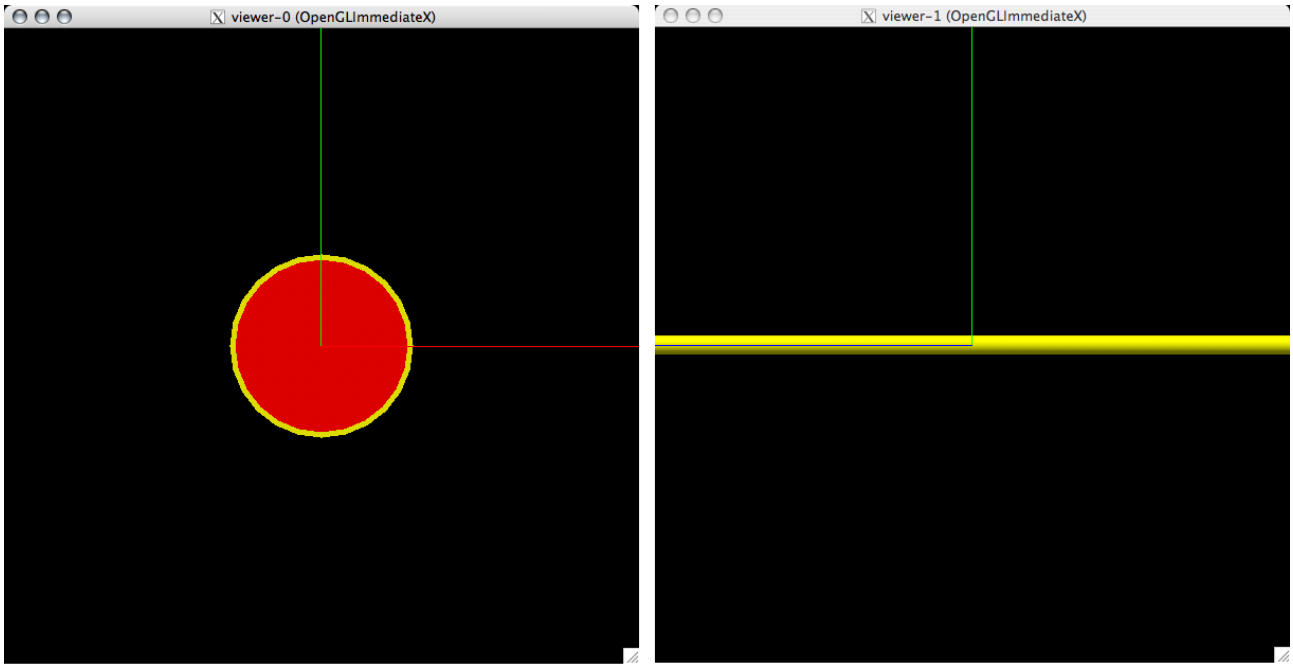


Figura 18 – Visualizzazioni del detector da due differenti angoli visuali, ottenute eseguendo il macro file vis.mac riportato nel Codice 5

Predisponendo opportunamente il *file* è possibile ottenere diverse visualizzazioni semplicemente spostando il simbolo #, che indica una linea di commento, piuttosto che specificare di volta in volta parametri diversi per i comandi: ad esempio la scelta del *driver* grafico da utilizzare (in questo caso è stato scelto il *driver* OPENGL), l'impostazione dell'angolo di visuale, con il comando `/vis/viewer/set/viewpointThetaPhi`, etc. In generale i comandi contenuti in `/vis/viewer` consentono di impostare i parametri per la visualizzazione che si vuole ottenere, ed è possibile chiaramente utilizzarli anche da linea di comando.

```
    #ifdef G4VIS_USE
        delete visManager;
    #endif

        delete runManager;
        return 0;
```

Codice 6 – Deallocazione degli oggetti vis Manager e runManager

Infine, come mostrato nel Codice 6, è necessario effettuare la *deallocazione* delle risorse.

5.2 Detector Construction

Le proprietà del *detector*, cioè le dimensioni, i materiali di cui è composto, la posizione e le zone sensibili (*Sensitive Detector*) sono definite nella classe `DetectorConstruction` che deriva la *mandatory user class* `G4VUserDetectorConstruction` del codice Geant4: essa contiene il metodo virtuale `Construct()` che deve essere implementato e che sarà invocato dal `runManager` (`G4RunManager`) durante l'inizializzazione dell'applicazione. All'interno di questo metodo deve essere inserito il codice per la definizione della geometria del *detector* nonché del volume "*world*". Il metodo dovrà restituire il puntatore all'istanza del volume fisico che rappresenta l'intero rivelatore. Il Codice 7 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();

    private:

    // Materials
           G4Material* air;
           G4Material* polystyrene;
           G4Material* plexiglass;
           G4Material* vacuum;

    // Global mother volume
           G4LogicalVolume * logicWorld;
           G4double halfWorldLength;

    // Detector Geometry
           G4VPhysicalVolume* physiFibraCladding;
           G4VPhysicalVolume* physiFibraCore;

    // Parameters for detector
```

```
            G4double fibraLen;
            G4double fibraRadius;
            G4ThreeVector posFibra;
};
```

Codice 7 – DetectorConstruction.hh, attributi e metodi della classe DetectorContruction

Prima di illustrare i dettagli della classe è opportuno un breve cenno agli aspetti che riguardano la creazione dei volumi componenti un rivelatore in Geant4.

Come già discusso nel terzo capitolo, la rappresentazione geometrica degli elementi che compongono un rivelatore si concentra sulla definizione di modelli solidi, sulla loro posizione spaziale nonché sulle loro relazioni logiche. Prima di ogni altra 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 (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 un'eccezione alla regola in quanto non può essere contenuto all'interno di nessun altro volume; è posizionato nell'origine del sistema di coordinate globale e rappresenta a sua volta il sistema di riferimento rispetto al quale sarà effettuato il posizionamento dei 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 un cubo con un lato di 10 centimetri, un cilindro di raggio 30 cm e lunghezza 75 cm);
- **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; altre informazioni contenute nell'istanza del volume logico riguardano gi attributi di visualizzazione e altri parametri definiti dall'utente relativi al tracciamento (ad esempio il campo magnetico o il puntatore ad un eventuale elemento sensibile);

- **fisico**: un “volume fisico” (`G4PVPlacement`) rappresenta il posizionamento spaziale di un volume logico. E’ 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. [15]

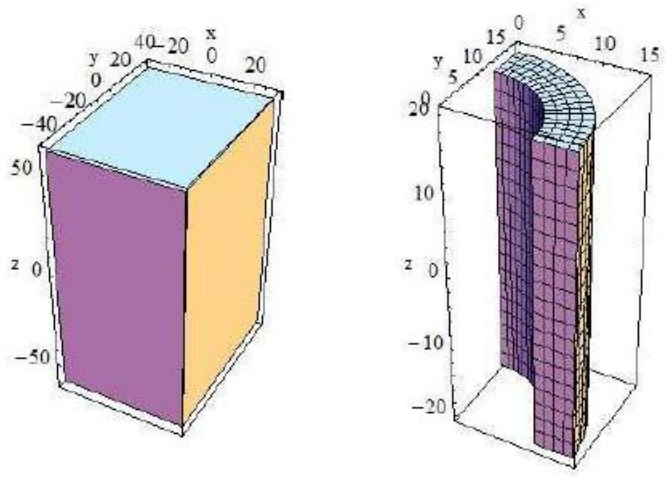


Figura 19 – Esempi di solidi: a sinistra `G4Box`; a destra `G4Tubs`.

Per capire meglio vediamo nel dettaglio come è stato costruito il volume *world* per l’applicazione, che nel caso specifico è rappresentato da un cubo di dimensioni 10x10x10 metri, riempito con il materiale “aria”. Dapprima è stato definito il “solido” istanziando un oggetto di classe `G4Box` all’interno del metodo `Construct()` della classe `DetectorConstruction`, specificando l’etichetta “*world*” e l’estensione lungo ciascuno degli assi cartesiani (Codice 8):

```
G4double halfWorldLength = 5.0*m;

G4Box * solidWorld= new G4Box("world",
                              halfWorldLength,
                              halfWorldLength,
                              halfWorldLength);
```

Codice 8 – Creazione del volume solido

Successivamente è stata creata l'istanza del volume logico partendo dall'oggetto "solido" appena creato e istanziando l'oggetto che rappresenta il materiale di cui il volume deve essere composto. Il materiale è rappresentato da un oggetto di classe `G4Material` che sarà passato come parametro al costruttore del volume logico.

Un materiale può essere definito a partire dagli elementi di cui è composto (`G4Element`) creando le molecole, oppure attingendo da un *database* di materiali predefiniti messo a disposizione dal *toolkit*. In questo caso è stato istanziato il materiale "aria", contenuto nel *database* dei materiali predefiniti, con cui è stato riempito il volume logico (Codice 9):

```
G4NistManager* man = G4NistManager::Instance();
G4Material* air = man->FindOrBuildMaterial("G4_AIR");

G4LogicalVolume* logicWorld= new G4LogicalVolume(solidWorld, air, "World");
```

Codice 9 – Definizione del materiale attraverso il database dei materiali e creazione del volume logico.

`solidWorld` è l'istanza del volume solido creato precedentemente, `air` è l'oggetto di classe `G4Material`, e "`World`" è l'etichetta assegnata al volume logico creato.

Dopo aver definito l'istanza del volume logico occorre posizionarlo creando l'oggetto volume fisico. Normalmente un tale oggetto riceve come parametri una matrice di rotazione, un vettore di traslazione e il volume madre all'interno del quale collocare il volume da posizionare: il volume *world* fa eccezione in quanto, si ribadisce, rappresenta il più grande volume creato e contiene tutti gli altri, pertanto non può essere contenuto all'interno di un altro; la sua istanza fisica deve essere creata come un oggetto `G4PVPlacement` al quale viene passato un *null pointer* come parametro relativo al volume logico madre. Inoltre non deve essere ruotato e deve essere posizionato nell'origine del sistema di coordinate globale.

```
G4VPhysicalVolume* physiWorld = new G4PVPlacement(0,           // no rotation
                                                  G4ThreeVector(), // at (0,0,0)
                                                  logicWorld,      // its logical volume
                                                  "World",         // its name
                                                  0,               // its mother  volume
                                                  false,           // no boolean operations
                                                  0);              // copy number
```

Codice 10 – esempio di creazione dell'oggetto "volume fisico"

Nel Codice 10 è mostrata la creazione dell'istanza `G4VPhysicalVolume` a cui è stata assegnata l'etichetta "`World`", che costituisce il posizionamento del volume logico `logicWorld` nell'origine del sistema di coordinate globali e senza rotazione (il primo parametro del costruttore è infatti un puntatore a *null*). In Figura 20 è riportata una visualizzazione del volume *world* creato che successivamente è stato reso invisibile attraverso il metodo `SetVisAttributes()` della classe `G4LogicalVolume` (Codice 11):

```
    //world volume is invisible
    logicWorld -> SetVisAttributes(G4VisAttributes::Invisible);
```

Codice 11 – Impostazione degli attributi di visualizzazione per il volume logico World

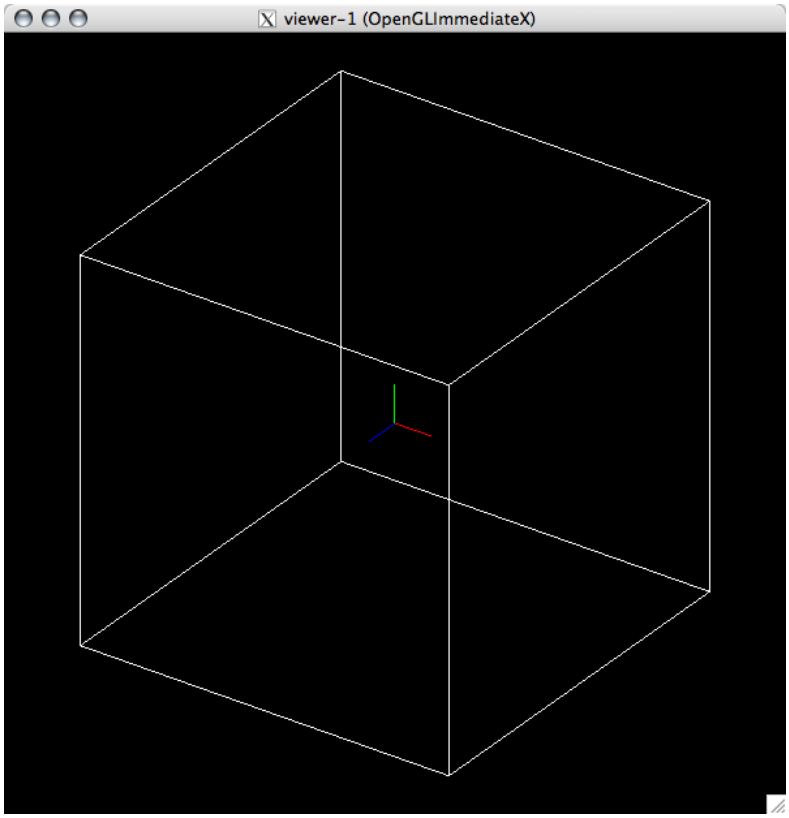


Figura 20 – Volume world definito per l'applicazione

5.2.1 Definizione della geometria della fibra

Il *detector*, posizionato nell'origine del sistema di riferimento, è costituito da una fibra dal diametro di 1 mm e lunga 120 cm. La fibra è composta da un *cladding* di *plexiglass* e da un *core* di polistirene. Oltre alla

definizione delle proprietà geometriche questa classe contiene 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.

Nel metodo `ComputeParameters()` della classe `DetectorConstruction`, riportato nel Codice 12, sono definite le dimensioni del volume *world* e dei volumi che compongono il *detector*.

```
void DetectorConstruction::ComputeParameters()
{
  // This function defines the defaults
  // of the geometry construction

  // ** world **
  halfWorldLength = 5.0*m;

  // ** Fibra **
  fibraLen = 1200*mm;
  fibraRadius = 0.5*mm;
  posFibra = G4ThreeVector(0,0,0);
}
```

Codice 12 – Metodo ComputeParameters() della classe DetectorConstruction.

Di seguito è riportato il codice del metodo `ConstructFibra()` (Codice 13): per prima cosa 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 che compongono il rivelatore, per rendere il *core* della fibra sensibile al passaggio delle particelle, si definisce il *sensitive detector*. Si crea l'oggetto `sensitive` della classe `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*.

```
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

      // MATRICE DI ROTAZIONE
      G4RotationMatrix* pRot = new G4RotationMatrix();
      //pRot->rotateX(0.*deg);
      //pRot->rotateY(0.*deg);
      pRot->rotateZ(0.*deg);

      //physiFibraCladding = new G4PVPlacement(0,         //no rotation
      physiFibraCladding = new G4PVPlacement( pRot,              // rotazione
                                       posFibra,             //translation
                                       fibraCladdingLogic,   //its logical volume
                                       "FibraCladding",      //its name
                                       logicWorld,           //its mother volume
                                       false,
                                       0);                   //copy number

      // *** CORE ***
      // 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 in phi
      // 2nd: define logical volume

      G4LogicalVolume* coreFibraLogic =
   new G4LogicalVolume( coreFibraSolid,     //its solid
                     polystyrene,          //its material
                     "coreFibraLogic");    //its name


      // 3rd: define the physical volume (= placed logical volume)

      physiFibraCore = new G4PVPlacement( 0,             //no rotation
                                  G4ThreeVector(0,0,0), //translation
                                  coreFibraLogic,       //its logical volume
                                  "physiFibraCore",     //its name
                                  fibraCladdingLogic,   //its mother volume
                                  false,
                                  1);                   //copy number

      G4Colour green(1,1,0);
      G4Colour red(1,0,0);

      coreFibraLogic->SetVisAttributes(new G4VisAttributes(red));
      fibraCladdingLogic->SetVisAttributes(new G4VisAttributes(green));

      /* SENSITIVE DETECTOR */

      // 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;
    }
```

Codice 13 – Metodo ConstructFibra per la definizione della geometria del rivelatore.

In Figura 21 è riportata la sezione della fibra vista nel piano XY (l'asse Z del sistema di riferimento coincide con l'asse della fibra).

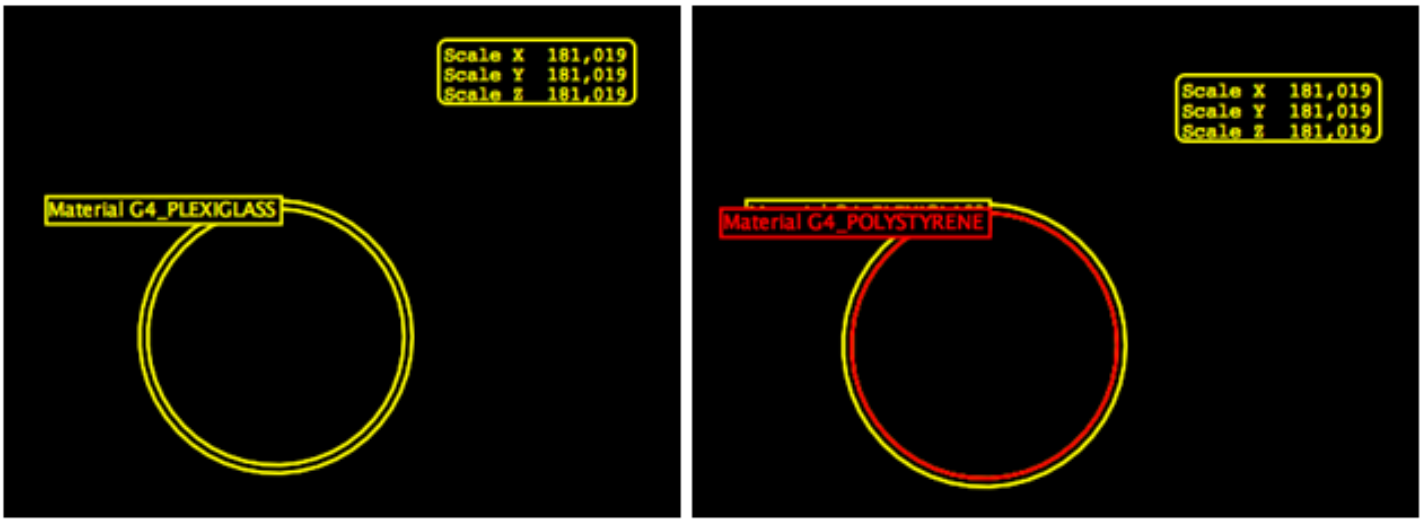


Figura 21 – Sezione della fibra vista nel piano XY

La creazione del volume *world*, di cui è stato fornito precedentemente un esempio dettagliato, avviene nel metodo `Construct()` riportato nel Codice 14. Il volume in questione è rappresentato da un cubo (`G4Box`) di dimensioni 10x10x10m (Figura 20) riempito con il materiale `air`. Il codice contiene inoltre l'invocazione al metodo `ConstructFibra()` per la costruzione del rivelatore appena discusso e riportato nel Codice 13.

```
    // World
    G4Box * solidWorld = new G4Box("world",
                                   halfWorldLength,
                                   halfWorldLength,
                                   halfWorldLength);

    logicWorld = new G4LogicalVolume( solidWorld, air, "World", 0, 0, 0);

    //  Must place the World Physical volume unrotated at (0,0,0).
    G4VPhysicalVolume * physiWorld = new G4PVPlacement(0,      // no rotation
                                            G4ThreeVector(), // at (0,0,0)
                                            logicWorld,             // its logical volume
                                            "World",         // its name
                                            0,               // its mother  volume
                                            false,           // no boolean operations
                                            0);              // copy number
    ConstructFibra();
```

Codice 14 – Definizione del volume world nel metodo Construct() della classe DetectorConstruction.

5.2.2 Materiali definiti

La classe `DetectorConstruction` contiene il metodo `DefineMaterials()`, illustrato di seguito nel Codice 15, in cui sono definiti i materiali componenti il rivelatore e l'ambiente circostante. Sono stati definiti 4 materiali facendo uso del *database* del *toolkit*: tramite l'oggetto "*Nist Manager*" (`G4NistManager`) è possibile recuperare il puntatore all'oggetto `G4Material` specificando il nome del materiale che si desidera definire e passandolo come parametro del metodo `FindOrBuildMaterial()` del `G4NistManager`.

```
  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");
  }
```

Codice 15 – Metodo DefineMaterials in cui sono definiti i materiali che compongono il detector.

Come accennato precedentemente, oltre alla possibilità di simulare i materiali a partire dal *database*, in Geant4 i materiali possono essere definiti come istanze di `G4Material` costruendo le molecole a partire dagli elementi che li compongono (oggetti della classe `G4Element` che descrivono le proprietà dell'atomo come il numero atomico, il numero di massa, etc.), o gli isotopi (`G4Isotope`) e specificando altre informazioni come la densità, la pressione, la temperatura, lo stato, etc. [15]

5.3 Physics List

Per quanto riguarda i processi fisici e le particelle che intervengono nelle simulazioni essi sono definiti all'interno della classe `PhysicsList` che deriva la *mandatory user class* `G4VUserPhysicsList`. 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.

Il Codice 16 riporta la struttura della classe `PhysicsList`:

```
class PhysicsList: public G4VUserPhysicsList
{
public:
    // Constructor
    PhysicsList();
    // Destructor
    ~PhysicsList();

protected:
    // Construct particle and physics (mandatory)

    // Construct particles
    void ConstructParticle();

    // Construct physics processes
    void ConstructProcess();

    // Define user cuts
    void SetCuts();

    void ConstructEM();
};
```

Codice 16 – PhysicsList.hh contiene i metodi e gli attributi della classe PhysicsList

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]

Tuttavia è possibile definire singolarmente le particelle, che devono essere istanze *singleton*, come mostra il Codice 17.

```
void PhysicsList::ConstructParticle()
{

  // pseudo-particles
  G4Geantino::Geantino();
  G4ChargedGeantino::ChargedGeantino();

  // particles
  G4Electron::Electron();
  G4Positron::Positron();
  G4Proton::Proton();

  // GAMMA
  G4Gamma::Gamma();
}
```

Codice 17 – 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 geometria del sistema e non interviene ai fini dell'esito delle simulazioni.

Per mantenere il codice ordinato e leggibile spesso si usa suddividere la registrazione di processi e particelle implementando dei metodi specifici per ciascuna categoria; in questo caso si è scelto di definire per i processi elettromagnetici, il metodo `ConstructEM()` che sarà richiamato all'interno del `ConstructProcess()`. Per quanto riguarda i processi fisici[p], sono stati registrati: effetto fotoelettrico, *Compton Scattering*, *Rayleigh Scattering* e *Gamma Conversion* per i gamma; *Coulomb Scattering*, ionizzazione e *Bremsstrahlung* per elettroni e positroni. Il metodo `ConstructEM()`, riportato nel Codice 18, registrerà i processi elencati.

```
void PhysicsList::ConstructEM()
{

    theParticleIterator->reset();

    while( (*theParticleIterator)() ){

    G4ParticleDefinition* particle = theParticleIterator->value();
    G4ProcessManager* pmanager = particle->GetProcessManager();
    G4String particleName = particle->GetParticleName();

    if (particleName == "gamma") {

           // PhotoElectric Effect
           pmanager->AddDiscreteProcess(new G4PhotoElectricEffect);

           // Compton Scattering
           pmanager->AddDiscreteProcess(new G4ComptonScattering);

           // Gamma Conversion
           pmanager->AddDiscreteProcess(new G4GammaConversion);

           // Rayleigh Scattering
```

[p] Per una descrizione più approfondita dei processi utilizzati si rimanda al capitolo 4.

```
            pmanager->AddDiscreteProcess(new G4RayleighScattering);

      } else if(particleName == "e+") {

            //      AddProcess riceve quattro parametri...
            //
            //      AddProcess( G4VProcess *aProcess,
            //                  G4int ordAtRestDoIt = ordInActive,
            //                  G4int ordAlongStepDoIt = ordInActive,
            //                  G4int ordPostStepDoIt = ordInActive)

            // Coulomb Scattering
            pmanager->AddProcess(new G4CoulombScattering);

            // Annihilation
            pmanager->AddProcess(new G4eplusAnnihilation);

            // Ionisation
            G4eIonisation* theIonisation = new G4eIonisation();
            pmanager->AddProcess(theIonisation,-1,1,1);

            // Bremsstrahlung
            G4eBremsstrahlung* theBremsstrahlung = new G4eBremsstrahlung();
            pmanager->AddProcess(theBremsstrahlung,-1,2,2);

      } else if(particleName == "e-") {

            // Ionisation
            G4eIonisation* theIonisation = new G4eIonisation();
            pmanager->AddProcess(theIonisation,-1,1,1);

            // Bremsstrahlung
            G4eBremsstrahlung* theBremsstrahlung = new G4eBremsstrahlung();
            pmanager->AddProcess(theBremsstrahlung,-1,2,2);

            // Coulomb Scattering
            pmanager->AddProcess(new G4CoulombScattering);
      }
    }
  }

  void PhysicsList::ConstructProcess()
  {
    AddTransportation();
    ConstructEM();
  }
```

Codice 18 – Metodi ConstructEM() e ConstructProcess() per la definizione dei processi elettromagnetici

Nel metodo è definito un oggetto *manager*, della classe `G4ProcessManager`, contenente una lista dei processi a cui ciascuna particella è sottoposta: attraverso questo gestore si legano i processi e le particelle definite all'interno del `ConstructParticle()`.

Attraverso l'oggetto `theParticleIterator` si scorre la lista delle particelle definite per l'applicazione e si invoca per ogni particella selezionata il metodo `AddProcess()` del *Process Manager* che aggiunge un processo alla lista. Il primo argomento rappresenta un puntatore al processo che si vuole aggiungere, mentre gli argomenti seguenti sono dei valori interi che rappresentano parametri di ordinamento del processo nel corrispettivo vettore dei processi del *Process Manager*; un parametro negativo indica che il

processo non sarà aggiunto nel vettore corrispondente. I tre argomenti si riferiscono rispettivamente all'ordinamento nei vettori `AtRestDoIt`, `AlongStepDoIt` e `PostStepDoIt`, che a loro volta corrispondono ai tre metodi virtuali della classe base `G4VProcess` da cui tutti i processi sono derivati e che descrivono il comportamento di un processo fisico.

Per alcuni processi sono definite solo azioni di tipo `AtRestDoIt`, `AlongStepDoIt` e `PostStepDoIt`. Essi sono definiti rispettivamente sulla base di particolari interfacce: `G4VAtRestProcess`, `G4VContinuousProcess` e `G4VDiscreteProcess`. La registrazione di questo tipo di processi avviene attraverso i metodi `AddAtRestProcess()`, `AddContinuousProcess()` e `AddDiscreteProcess()`. Quest'ultimo è il metodo utilizzato nel Codice 18 per i processi elettromagnetici che riguardano le interazioni dei raggi gamma con la materia, i quali entrano in gioco alla fine di uno *step* e pertanto prevedono solo azioni di tipo `PostStepDoIt`; le classi che descrivono tali processi sono implementate sulla base dell'interfaccia `G4VDiscreteProcess`. Il metodo `AddDiscreteProcess()` potrà ricevere due argomenti, di cui il primo sarà il puntatore al processo che si desidera aggiungere e il secondo un intero per l'ordinamento nel vettore corrispondente. Se il parametro di ordinamento non viene specificato, come nel caso del Codice 18, il processo verrà aggiunto alla fine della lista del vettore dei processi. Per ulteriori approfondimenti si rimanda al *Software Reference Manual* [17] e al *Geant4 User's Guide for Application Developer* [15].

Fa eccezione il processo di trasporto che descrive il moto delle particelle e viene registrato per tutte le particelle nel `ConstructProcess()` attraverso il metodo `AddTransportation()` della classe `G4VUserPhysicsList`.

Il terzo metodo che occorre definire è il `SetCuts()` in cui devono essere impostati i valori di *cut-off* per le particelle; in questo caso sono stati scelti quelli di *default* per tutte le particelle attraverso il metodo `SetCutsWithDefault()` della classe `G4VUserPhysicsList`, che usa il valore della variabile `defaultCutValue` della classe, impostato a 1.0 mm (Codice 19).

```
void PhysicsList::SetCuts()
{
  //G4VUserPhysicsList::SetCutsWithDefault method sets
  //the default cut value for all particle types
  SetCutsWithDefault();
}
```

Codice 19 – Metodo SetCuts() per impostare i valori di cut-off per le particelle.

5.4 Primary Generator Action

Questa classe, implementata derivando la *mandatory user class* del codice Geant4 `G4VUserPrimaryGeneratorAction`, 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.

Queste caratteristiche sono racchiuse nell'oggetto `G4ParticleGun`, che rappresenta l'effettivo generatore di particelle primarie, istanziato nel costruttore.

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()` del `G4ParticleGun`. [15]

```
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);

           // get methods
           G4String getSourceType();
           G4String getDistributionType();
           G4double getDistanceY();

    private:
```

```
            G4ParticleGun* gun;

            PrimaryGeneratorMessenger* gunMessenger;
            G4double distanceY;
            G4String distribType;
            G4ThreeVector direct;
            G4String source;
};
```

Codice 20 – Metodi e attributi della classe PrimaryGeneratorAction

Gli attributi relativi alla sorgente (`distanceY`, `distribType`, `source`) sono inizializzati nel costruttore (Codice 21) con dei valori di *default* che potranno essere cambiati interattivamente attraverso alcuni comandi definiti nella classe `PrimaryGeneratorMessenger` descritta più avanti. Tali comandi effettueranno le chiamate ai metodi `setSource()`, `setDistanceY()` e `setDistributionType()` implementati per settare i valori delle variabili `source`, `distanceY` e `distribType` che saranno passati come parametri ai metodi del *particle gun* per impostare le caratteristiche dell'evento primario.

```
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");
}
```

Codice 21 – 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()` che genererà l'evento primario. La classe contiene poi i due metodi `GenerateIsotropicFlux()` e `DistributionUniformInSolidAngle()` 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 22 è riportata a sinistra una vista parziale della fibra nel piano YZ e a destra la stessa sezione con un fascio di raggi gamma sparati in maniera isotropica nella porzione di angolo solido coperta dalla fibra.

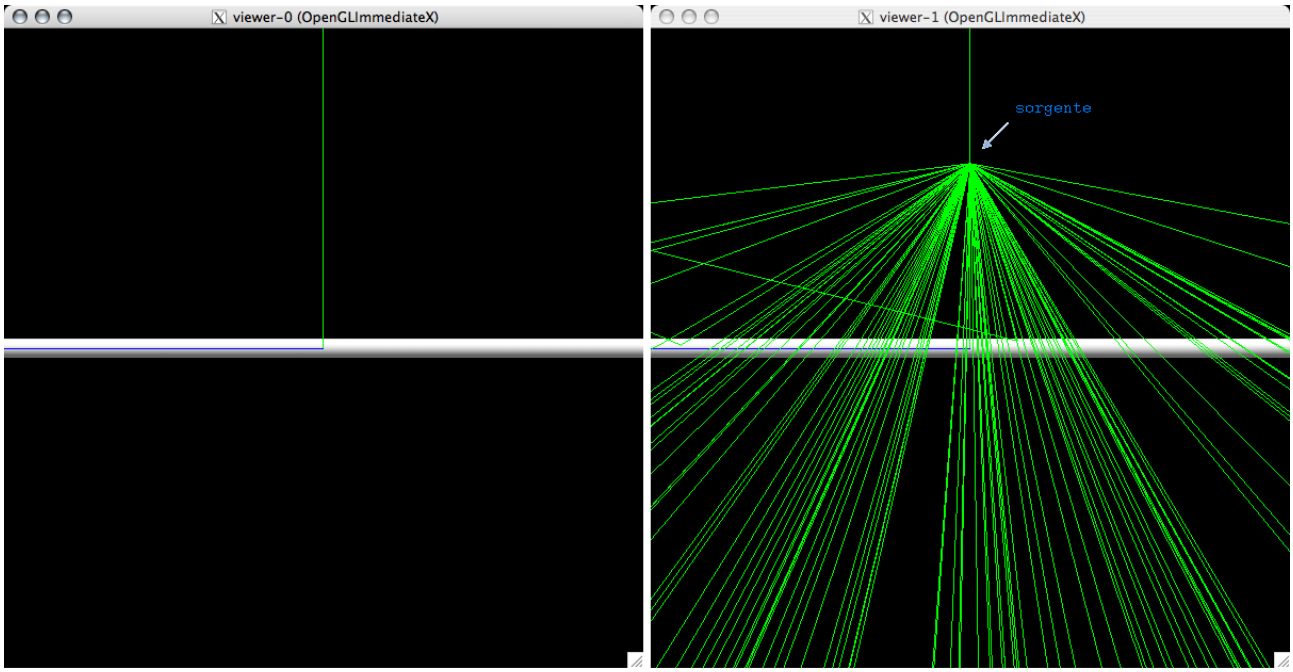


Figura 22 - Fibra vista nel piano YZ (a sinistra) e con un fascio di raggi gamma sparati dalla sorgente nell'angolo solido (a destra)

Di seguito (Codice 22) sono riportati i due metodi per la generazione dei due tipi di flusso previsti per le simulazioni (`GenerateIsotropicFlux()` e `DistributionUniformInSolidAngle()`).

```
G4ThreeVector PrimaryGeneratorAction::GenerateIsotropicFlux()
{
    //distribution uniform in 4pigreco

       G4double cosTheta = 2*G4UniformRand() - 1., phi = twopi*G4UniformRand();
       G4double sinTheta = std::sqrt(1. - cosTheta*cosTheta);

       G4double ux = sinTheta*std::cos(phi),
                uy = sinTheta*std::sin(phi),
                uz = cosTheta;
       return G4ThreeVector(ux,uy,uz);
}

G4ThreeVector PrimaryGeneratorAction::DistributionUniformInSolidAngle()
{
    // distribution uniform in fiber solid angle

    // source distance
         G4double d = distanceY;
    // distanceY : distance between source and detector

    // fiber radius
         G4double r = 0.5*mm;
    // fiber lenght /2
         G4double halfZ = 600*mm;
    // hypotenuse
```

```
            G4double i;

      // *** THETA

      // Theta varia tra -delta e -pigreca+delta
            i = std::sqrt((halfZ*halfZ)+(d*d));

      G4double delta = std::acos(halfZ/i);

      // estremo superiore di cosTheta = cos(-delta) = cos(delta)
      G4double a = std::cos(delta);

      // cosTheta varia tra -a e a
      G4double cosTheta = (2*a)*G4UniformRand() - a;

      // *** PHI

      //ipotenusa
      i = std::sqrt((r*r)+(d*d));

      G4double beta = std::asin(d/i);

      // phi varia tra [- beta; -pigreco + beta]

      G4double phi = (- beta) + (-pi + 2*beta) * G4UniformRand();
      G4double sinTheta = std::sqrt(1. - cosTheta*cosTheta);

      // passaggio a coordinate cartesiane

      G4double ux = sinTheta*std::cos(phi),
               uy = sinTheta*std::sin(phi),
               uz = cosTheta;


      return G4ThreeVector(ux,uy,uz);
   }
```

Codice 22 – Metodi GenerateIsotropicFlux e DistributionUniformInSolidAngle della classe PrimaryGeneratorAction.

I due metodi fanno uso della funzione `G4UniformRand()` delle librerie CLHEP, che restituisce un numero casuale sulla base di una distribuzione uniforme nell'intervallo (0,1); tramite questa funzione è stato possibile ottenere una variabile casuale distribuita uniformemente in un intervallo a piacere variando gli estremi a seconda delle necessità, al fine di ottenere delle coordinate polari casuali all'interno dell'area desiderata, convertite successivamente in coordinate cartesiane, per impostare la distribuzione delle particelle generate attraverso il metodo `SetParticleMomentumDirection()`, che riceve come parametro un oggetto `G4ThreeVector` inizializzato con i tre valori *random* ottenuti.

Nel Codice 23 è riportato il metodo principale della classe `PrimaryGenertorAction`, il `GeneratePrimaries()`, invocato all'inizio di ogni evento. All'interno del metodo, dopo l'invocazione dei metodi del *particle gun* per impostare le caratteristiche desiderate, viene invocato il metodo `generatePrimaryVertex()` che darà origine all'evento primario. Il codice contenuto nel metodo sarà eseguito un numero di volte pari al numero di

eventi che si sceglie di generare in fase di esecuzione del programma in seguito al lancio del comando `/run/beamOn N` dove N è il numero di eventi.

```
void PrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{
    if( source == "Co60") {
            direct = setDistributionType(distribType);

            gun->SetParticleMomentumDirection(direct);
            gun->SetParticleEnergy(1.33*MeV);
            gun->GeneratePrimaryVertex(anEvent);

            direct = setDistributionType(distribType);

            gun->SetParticleMomentumDirection(direct);
            gun->SetParticleEnergy(1.17*MeV);
            gun->GeneratePrimaryVertex(anEvent);

    } else if (source == "Cs137") {
            direct = setDistributionType(distribType);

            gun->SetParticleMomentumDirection(direct);
            gun->SetParticleEnergy(662*keV);
            gun->GeneratePrimaryVertex(anEvent);
    }
}
```

Codice 23 – Metodo GeneratePrimaries() della classe PrimaryGeneratorAction

5.5 Fibra Sensitive Detector

Per definire la risposta del rivelatore occorre implementare una classe che gestisca una collezione di oggetti, detti *hit*, ognuno dei quali rappresenta un'istantanea dell'interazione fisica di una traccia nella regione sensibile del *detector*.

La classe in questione, chiamata `FibraSensitiveDetector`, deriva la classe astratta del codice Geant4 `G4VSensitiveDetector` e ne ridefinisce i metodi `Initialize()`, `ProcessHits()`, `EndOfEvent()`, attraverso i quali avviene la gestione degli *hit* e della *hit collection*. Prima di analizzare i tre metodi elencati occorre fornire qualche dettaglio sulle classi che definiscono gli *hit* e la *hit collection*.

Gli oggetti *hit* vengono definiti sulla base della classe astratta `G4VHit` del codice Geant4 che è stata derivata per implementare la classe `FibraHit`. 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* per elaborare tali informazioni in un secondo momento.

La *hit collection* definita per l'applicazione nella classe `FibraSensitiveDetector` 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*. Per definire la *hit collection* si inizia assegnando (all'interno della classe `FibraHit`) il nuovo nome `FibraHitCollection` al tipo di dato `G4THitsCollection`, così da non creare confusione (Codice 24).

```
#include "G4THitsCollection.hh"

... ... ...

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

Codice 24 – Classe FibraHit: assegnazione del nome FibraHitCollection al tipo di dato G4THitsCollection.

In tal modo è stato anche specificato 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`.

```
class FibraSensitiveDetector : public G4VSensitiveDetector
{
    public:
      // Constructor
            FibraSensitiveDetector(G4String SDname);
      // Destructor
            ~FibraSensitiveDetector();
    public:
            G4int hitCount;
            G4int spettro[bin];
            G4double eneTotRun,eneTotEvento;

      // Methods from base class G4VSensitiveDetector
      // Mandatory base class method : it must to be overloaded:
            G4bool ProcessHits(G4Step *step, G4TouchableHistory *ROhist);
      // (optional) method of base class G4VSensitiveDetector
            void Initialize(G4HCofThisEvent* HCE);
      // (optional) method of base class G4VSensitiveDetector
            void EndOfEvent(G4HCofThisEvent* HCE);

      // other methods
            void initSpettro();
            G4double eneTot();
            void scrivi_su_file(G4String nomeFile);
            void setSpettro(G4double energy);
    private:
            FibraHitCollection* hitCollection;
};
```

Codice 25 – Attributi e metodi della classe FibraSensitiveDetector

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 `collectionName`, attributo della classe `G4VSensitiveDetector`. Quanto descritto è riportato nel Codice 26:

```
FibraSensitiveDetector::FibraSensitiveDetector(G4String SDname)
: G4VSensitiveDetector(SDname)
{
  G4String myCollectionName = "FibraHitCollection";
  collectionName.insert(myCollectionName);

  hitCount = 0;
  eneTotRun = eneTotEvento = 0;
  initSpettro();
}
```

Codice 26 – Metodo costruttore della classe FibraSensitiveDetector

Successivamente nel metodo `Initialize()` viene istanziato l'oggetto vero e proprio, come mostra il Codice 27. Il costruttore della *hit collection* riceve due parametri, di cui il primo è il nome assegnato al *sensitive detector* e il secondo è il nome assegnato alla *hit collection*.

```
// Create the hit collection, remember the hit collection constructor wants two
// strings: the name of the SD and the name of the collection:

    hitCollection = new FibraHitCollection( GetName(), collectionName[0] );
```

Codice 27 – Metodo Initialize() di FibraSensitiveDetector.

Sempre all'interno del metodo `Initialize()` occorre aggiungere la *hit collection* appena creata alla lista delle collezioni dell'evento che si sta considerando. Ciascun oggetto "evento" della classe `G4Event` possiede un oggetto di classe `G4HCofThisEvent` che è un contenitore di collezioni di *hit*. Le *hit collection* definite vengono collezionate attraverso i loro puntatori in questo contenitore, per ogni evento. Ciò avviene attraverso il metodo `AddHitsCollection()` di `G4HCofThisEvent` che riceve come parametri l'ID della collezione e il puntatore alla collezione (Codice 28).

```
static G4int HCID = -1;
if (HCID<0) HCID = GetCollectionID(0);
HCE->AddHitsCollection(HCID, hitCollection);
```

Codice 28 – Metodo Initialize(): aggiunta della hit collection al contenitore HCE (Hit Collection of This Event) dell'evento corrente.

La gestione della collezione di *hit* avverrà attraverso la classe `FibraSensitiveDetector` che selezionerà gli oggetti *hit* da collezionare attraverso il metodo `ProcessHits()` riportato nel Codice 29.

```
G4bool FibraSensitiveDetector::ProcessHits(G4Step *step, G4TouchableHistory *)
{
    G4TouchableHandle touchable = step->GetPreStepPoint()->GetTouchableHandle();
    G4int copyNo = touchable->GetVolume(0)->GetCopyNo();
    G4String name = touchable->GetVolume(0)->GetName();

    G4String detectorName = "physiFibraCore";
    G4Track* track = step->GetTrack();
    G4StepPoint* preStepPoint = step->GetPreStepPoint();

    G4double edep = step->GetTotalEnergyDeposit();

    if((track->GetParentID() == 0))
    {
     if((preStepPoint->GetStepStatus() == fGeomBoundary) && (name == detectorName))

           {
                  FibraHit* aHit = new FibraHit(copyNo);
                  aHit->AddEdep(edep);

                  // aggiungo il nuovo hit
                  hitCollection->insert(aHit);
           }
    }

    eneTotRun += edep;
    eneTotEvento += edep;
    return true;
}
```

Codice 29 – Metodo ProcessHits() della classe FibraSensitiveDetector.

Tale metodo viene richiamato dallo `G4SteppingManager` ogni volta che una particella compie uno *step* all'interno del volume logico al quale il *sensitive detector* fa riferimento, in questo caso il *core* della fibra. L'implementazione del metodo ha lo scopo di contare gli *hit* delle particelle primarie e registrare l'energia totale rilasciata nel volume, evento per evento.

Per contare solamente il numero di particelle primarie che incidono nel *sensitive detector* si era inizialmente deciso di usare il metodo `IsFirstStepInVolume()` della classe `G4Step` del codice, che avrebbe dovuto restituire il valore di un *flag* che, come dice il nome del metodo stesso, segnala se lo *step* corrente della particella è il primo nel volume: in realtà nel corso dell'implementazione si è visto che questo metodo non è implementato nel codice Geant4 o comunque non restituisce un valore attendibile. In seguito a ciò si è deciso di procedere diversamente, mediante l'utilizzo di altri *flag*, come spiegato di seguito.

Come si può vedere nel Codice 29, per contare gli *hit* delle particelle primarie relativi al loro primo attraversamento del volume sensibile è stato fatto dapprima un confronto tra l'attributo `ParentID` della

traccia e il valore 0 al fine di prendere in considerazione solo le particelle primarie: infatti se il `ParentID` dell'oggetto `G4Track` relativo alla traccia corrente è pari a 0, ciò significa che quella che ha compiuto lo *step* corrente è una particella primaria (un gamma) e quindi si potrebbe essere interessati a conteggiarla; a questo punto essendo vera la condizione, viene eseguito il secondo confronto che fa uso del *flag* `fGeomBoundary`, il quale sarà settato a 1 se le coordinate dello *step* precedente a questo (ottenute con il metodo `GetPreStepPoint()` dell'oggetto `step` e contenute nella variabile `preStepPoint`) fanno riferimento ad una linea di confine del volume; allo stesso tempo si deve verificare che il volume corrente sia il *detector*; ciò è fatto attraverso un confronto tra la variabile `name` e una stringa precedentemente inizializzata con il nome del *detector*. Occorre notare che questo è possibile in quanto lo *stepping* di una particella in Geant4 prevede che l'attraversamento di una linea di confine tra un volume e un altro corrisponda a due *step* separati che hanno un punto in comune sulla linea di confine.

Se tutte le condizioni poste risultano vere la particella in questione è un gamma che entra per la prima volta nel volume sensibile e deve essere conteggiato. A tal fine si istanzia un oggetto di classe `FibraHit` e lo si inserisce nella `hitCollection`.

Il conteggio totale sarà dato dalla dimensione della *hit collection*, che alla fine di ogni evento (nel metodo `EndOfEvent()`) viene sommata alla variabile `hitCount`.

L'energia rilasciata da una particella nel *sensitive detector* può essere ricavata attraverso il metodo apposito `GetTotalEnergyDeposit()` della classe `G4Step` del codice Geant4; l'energia totale dell'evento memorizzata nella variabile `eneTotEvento`, ottenuta sommando l'energia depositata ad ogni *hit*, viene usata per ricavare lo spettro dell'energia accumulata nella fibra, evento per evento, nel corso del *run*. Queste informazioni vengono salvate in un file di testo per essere elaborate in un secondo momento. Tale *file* denominato "`spettroX.txt`" (dove X sarà l'ID del *run*) viene salvato attraverso il metodo `scrivi_su_file()` invocato alla fine del *run* dalla classe `RunAction` descritta in seguito.

5.6 Primary Generator Messenger

Come già discusso nel paragrafo 5.1, 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. [15] La classe *messenger* sviluppata per l'applicazione è la `PrimaryGeneratorMessenger`, che contiene la definizione degli oggetti per i comandi 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*. Il codice mette a disposizione diverse tipologie di classi variabili sulla base dei parametri che il comando può ricevere: come si può vedere nel Codice 30 le classi in questione utilizzate nell'applicazione sono `G4UIcmdWithAString` e `G4UIcmdWithADoubleAndUnit`.

```
class PrimaryGeneratorMessenger: public G4UImessenger
{
    public:
      PrimaryGeneratorMessenger(PrimaryGeneratorAction*);
      virtual ~PrimaryGeneratorMessenger();

      void SetNewValue(G4UIcommand*, G4String);

    private:
           PrimaryGeneratorAction*             action;
           G4UIdirectory*                      gunDir;
           G4UIcmdWithAString*                 distrCmd;
           G4UIcmdWithAString*                 sourceType;
           G4UIcmdWithADoubleAndUnit*          sourceCmd;
};
```

Codice 30 - Metodi e attributi della classe PrimaryGeneratorMessenger.

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, riportati nel Codice 30, sono: `distrCmd`, `sourceCmd`, `sourceType`, a cui corrispondono i comandi: `setDistrib`,

`setDistance`, `setSourceType`. Come si intuisce dai loro nomi i tre comandi appena elencati 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 from the sensitive
detector.");
        sourceCmd->SetGuidance("  Choice : Double and Unit");
        sourceCmd->SetParameterName("Distance",true);
        sourceCmd->SetDefaultValue(1.0);
        sourceCmd->SetDefaultUnit("cm");
        sourceCmd->AvailableForStates(G4State_PreInit,G4State_Idle);

        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);

    }
```

Codice 31 – Metodo costruttore di PrimaryGeneratorMessenger

Per registrare i nuovi comandi occorre istanziare l'oggetto *messenger* nel costruttore della `PrimaryGeneratorAction`, come riportato nel Codice 32.

```
PrimaryGeneratorAction::PrimaryGeneratorAction()
{
    // create a messenger for this class
    gunMessenger = new PrimaryGeneratorMessenger(this);

    ... ... ...
}
```

Codice 32 - Metodo costruttore della classe PrimaryGeneratorAction

Il costruttore della classe `PrimaryGeneratorMessenger` riceve un parametro che corrisponde al puntatore all'istanza del `PrimaryGeneratorAction` che servirà per richiamare i metodi contenenti il codice da associare ai comandi, all'interno del `SetNewValue()` (Codice 33).

```
void PrimaryGeneratorMessenger::SetNewValue(G4UIcommand* command, G4String newValue)
{
    if( command == distrCmd ) {
            action->setDistributionType(newValue);
    } else if (command == sourceCmd ) {
            G4double value = sourceCmd->GetNewDoubleValue(newValue);

            action->setDistanceY(value);

    } else if (command == sourceType) {
            action->setSource(newValue);
    }
}
```

Codice 33 – Metodo SetNewValue() di PrimaryGeneratorMessenger

5.7 Run Action

La classe `RunAction` è implementata derivando la classe virtuale `G4UserRunAction` del codice che fa parte della categoria delle *optional user classes* che il *toolkit* mette a disposizione, i cui metodi devono essere riscritti per guadagnare il controllo della simulazione a vari stadi.

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*[q];

[q] La *physics table* è una tabella contenente valori utili alla simulazione come *cross-section*, *energy loss*, etc. calcolati e tabulati prima dell'inizio del ciclo di eventi attraverso il metodo `BuildPhysicstable()` della classe `G4VProcess` del codice.

- `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. [15]

```
class RunAction : public G4UserRunAction
{
    public:
            // constructor
            RunAction(PrimaryGeneratorAction*);

            // destructor
            virtual ~RunAction() {};

            // Called at the beginning of each run
            void BeginOfRunAction(const G4Run*);

            // Called at the end of each run
            void EndOfRunAction(const G4Run*);

    private:
            PrimaryGeneratorAction*      action;
};
```

Codice 34 - Metodi e attributi della classe RunAction

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();
}
```

Codice 35 – Metodo BeginOfRunAction() della classe RunAction

Il Codice 35 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. Tali variabili, che

rappresentano gli attributi del *sensitive detector*, possono essere aggiornate ottenendo il puntatore all'oggetto 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()` (Codice 36) eseguito alla fine di ogni *run* contiene il codice per salvare le informazioni relative al *run* appena eseguito in un *file* di testo a cui è assegnato il nome "*output*" 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;
    outFile << "\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);
}
```

Codice 36 – Metodo EndOfRunAction() della classe RunAction

5.8 Variabili d'ambiente e compilazione

Dopo aver implementato il codice di tutte le classi necessarie l'eseguibile viene generato attraverso il meccanismo *GNUmake* controllato da alcuni *script* (`architecture.gmk`, `common.gmk`, `globlib.gmk`, `binmake.gmk`, `GNUmakefile`) 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*" (valido per MacOS) 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=/Applications/g4work
export DYLD_LIBRARY_PATH=/Applications/CLHEP/lib
source /Applications/geant4.9.3.p02/env.sh
```

Codice 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 (che nel caso del Codice 37 è la *directory Applications*), `DYLD_LIBRARY_PATH` deve contenere il percorso alle librerie CLHEP che viene stabilito durante la fase di installazione del *toolkit*; l'ultima riga esegue lo *script* `env.sh` fornito dal *toolkit*, che imposta tutte le altre variabili d'ambiente necessarie.

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`. [15]

6 Simulazioni eseguite e analisi dei dati ottenuti

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

6.1 Verifiche effettuate

Prima di iniziare le simulazioni si è proceduto con la verifica della geometria del sistema. A tal fine 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 le simulazioni in modalità grafica è stato usato il *macro file* `vis.mac` (il cui contenuto è stato riportato e discusso nel paragrafo 5.1 del capitolo precedente) lanciato attraverso il comando `control/execute vis.mac`, che genera due visualizzazioni della geometria da due differenti punti di vista. Le simulazioni di test in modalità grafica sono state eseguite generando pochi eventi per non sovraccaricare le risorse del sistema, già notevolmente rallentate a causa della visualizzazione, che altrimenti richiederebbero troppo tempo, peraltro inutilmente, per completare la simulazione.

Per verificare l'efficienza geometrica è stato calcolato il valore dell'angolo solido del rivelatore visto dalla sorgente. Per definizione l'angolo solido 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. (Figura 23)

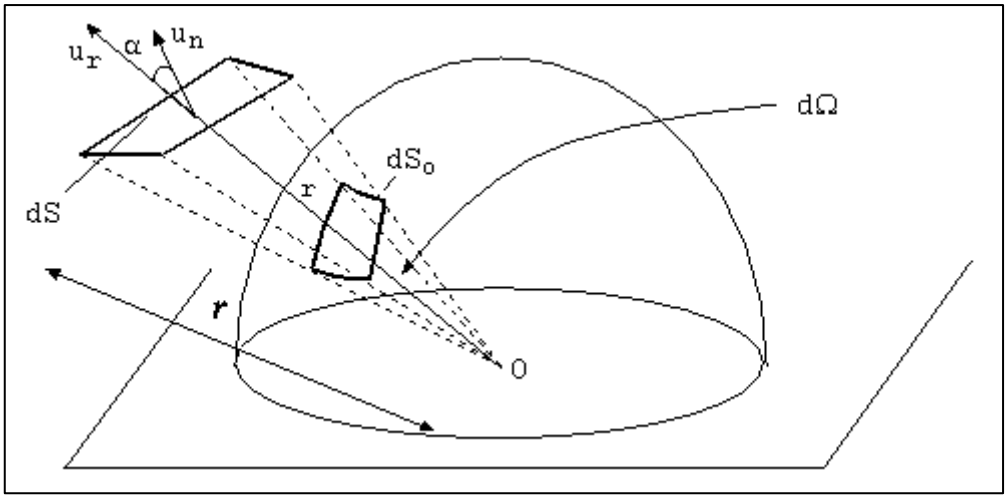


Figura 23 – Definizione di angolo solido.

Si definisce angolo solido infinitesimo la quantità

$$d\Omega = \frac{\mathrm{dS}\cos\alpha}{\mathrm{r}^2} = \frac{\mathrm{dS}_0}{\mathrm{r}^2}$$

la superficie dS_0 rappresenta un elemento di superficie proiettato sulla sfera unitaria. [18]

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:

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

dove $\hat{\boldsymbol{n}}$ 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: [19]

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

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 $\theta_0 = 0$ 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 ε_g, sarà pari a:

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

Dove A_{riv} e Ω sono l'area e l'angolo solido del rivelatore e Ω_{tot} l'angolo solido totale.

Nel caso di un rivelatore di dimensioni finite l'espressione va estesa tramite l'integrale:

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

Dove Ω 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. Per il calcolo analitico della porzione di angolo solido si è fatto riferimento alla figura seguente (Figura 24):

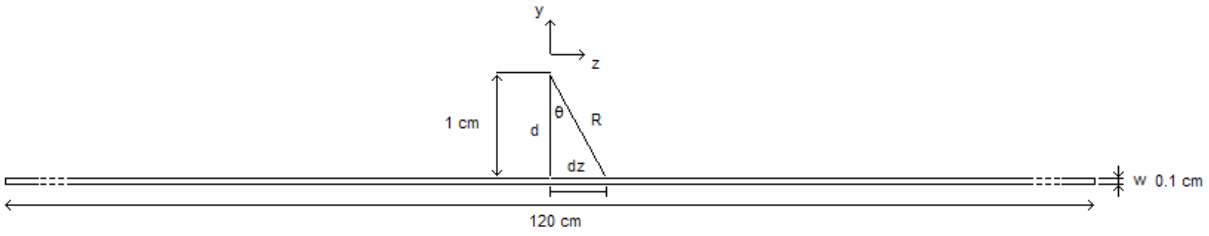


Figura 24 – Immagine di riferimento per il calcolo analitico dell'angolo solido

$$R = \frac{d}{\cos\theta}$$

$$z = R\sin\theta = d\tan\theta$$

$$dz = \frac{d}{cos^2\theta}\,d\theta$$

$$d\Omega = \frac{w\cos\theta\,dz}{R^2} = w\frac{cos^2\theta}{d^2}\cos\theta\frac{d}{cos^2\theta}d\theta = \frac{w}{d}\cos\theta\,d\theta$$

$$\Omega = \int_{-\frac{\pi}{2}}^{\frac{\pi}{2}}\frac{w}{d}\cos\theta\,d\theta = \frac{w}{d}\left[\sin\frac{\pi}{2} - \sin(-\frac{\pi}{2})\right] = \frac{2w}{d}$$

Dalla definizione di efficienza geometrica si ricava infine:

$$\varepsilon_g = \frac{\Omega}{4\pi} = \frac{2w}{d}\frac{1}{4\pi} = \frac{w}{2\pi d}$$

Sostituendo i valori si ottiene il valore della porzione di angolo solido visto dalla sorgente alla distanza di 1 cm, come mostrato in Figura 24, quindi dell'efficienza geometrica:

$$\varepsilon_g = \frac{0.1\ cm}{2\pi(1.0\ cm)} = 0{,}015923$$

Questo valore teorico è stato confrontato con il valore ottenuto effettuando una simulazione di test irradiando 2.000.000 di *geantini* in maniera isotropica nello spazio; il numero delle particelle incidenti sul rivelatore ottenuto è stato rapportato al numero totale di particelle generate; tale rapporto è risultato consistente con il valore teorico calcolato:

$$\varepsilon_g = \frac{N_H}{N_{TOT}} = \frac{30163}{2000000} = 0{,}0150815$$

anche in considerazione del fatto che per la simulazione si è adoperato lo spessore efficace di 0.94 cm che rappresenta la parte sensibile della fibra. Nel corso delle simulazioni con le particelle gamma, questi stessi valori sono stati calcolati al variare della distanza della sorgente dalla fibra e in tutti i casi i valori ottenuti tramite le simulazioni sono rimasti coerenti con quanto calcolato analiticamente.

6.2 *Setup* delle simulazioni

Per le simulazioni sono stati presi in considerazione due parametri: il tipo di distribuzione e la distanza della sorgente dal volume; sono stati decisi 13 valori di distanze, che variano da 1 mm a 25 mm, ed eseguite due serie di *run* per ognuno dei quali sono stati lanciati 1.000.000 di eventi variando tra un *run* e l'altro il valore della distanza; la prima serie generando le particelle con distribuzione isotropica in tutto lo spazio impostando i parametri a *runtime* con il *macro file* `Co60-iso.mac` riportato nel Codice 38, e la seconda impostando la distribuzione isotropica nell'angolo solido del *detector* attraverso l'esecuzione del *macro file* `Co60-ang.mac` (Codice 39).

```
# source type:     Co60 (default) [2 gammas per event: 1.17 MeV, 1.33 MeV]
# distrib type:    iso (default)

# run 0
/Fibra/setDistance 1.0 mm
/run/beamOn 1000000
# run 1
/Fibra/setDistance 2.0 mm
/run/beamOn 1000000
# run 2
/Fibra/setDistance 3.0 mm
/run/beamOn 1000000
# run 3
/Fibra/setDistance 4.0 mm
/run/beamOn 1000000
# run 4
/Fibra/setDistance 5.0 mm
/run/beamOn 1000000
# run 5
/Fibra/setDistance 6.0 mm
/run/beamOn 1000000
# run 6
/Fibra/setDistance 7.0 mm
/run/beamOn 1000000
# run 7
/Fibra/setDistance 8.0 mm
/run/beamOn 1000000
# run 8
/Fibra/setDistance 9.0 mm
/run/beamOn 1000000
# run 9
/Fibra/setDistance 1.0 cm
/run/beamOn 1000000
# run 10
/Fibra/setDistance 1.5 cm
/run/beamOn 1000000
# run 11
/Fibra/setDistance 2.0 cm
/run/beamOn 1000000
# run 12
/Fibra/setDistance 2.5 cm
/run/beamOn 1000000
```

Codice 38 – Macro file Co60-iso.mac per impostare i parametri delle simulazioni relativamente alla distribuzione isotropica in tutto lo spazio.

Lo *script* lancia 13 simulazioni, una dietro l'altra, impostando tra un *run* e l'altro un diverso valore della distanza della sorgente attraverso il comando `setDistance` (definito nella classe *messenger* descritta nel capitolo precedente) che riceve come parametri il valore della distanza e l'unità di misura. Il tipo di distribuzione è stabilito all'inizio dello *script* mediante il comando `setDistrib` (anch'esso definito nella classe *messenger*): il comando serve ad impostare uno tra i due tipi di distribuzione previsti attraverso le stringhe "`solidAngle`" oppure "`iso`": la prima fa riferimento alla distribuzione isotropica nell'angolo solido, mentre la seconda si riferisce alla distribuzione isotropica in tutto lo spazio. Dopo l'esecuzione delle 24 simulazioni sono stati raccolti in tabelle i dati presenti nel *file* di *output* generati e sono stati tracciati i grafici di interesse.

```
# source type:    Co60 (default) [2 gammas per event: 1.17 MeV, 1.33 MeV]
# distrib type:   solidAngle

/Fibra/setDistrib solidAngle

# run 0
/Fibra/setDistance 1.0 mm
/run/beamOn 1000000
# run 1
/Fibra/setDistance 2.0 mm
/run/beamOn 1000000
# run 2
/Fibra/setDistance 3.0 mm
/run/beamOn 1000000
# run 3
/Fibra/setDistance 4.0 mm
/run/beamOn 1000000
# run 4
/Fibra/setDistance 5.0 mm
/run/beamOn 1000000
# run 5
/Fibra/setDistance 6.0 mm
/run/beamOn 1000000
# run 6
/Fibra/setDistance 7.0 mm
/run/beamOn 1000000
# run 7
/Fibra/setDistance 8.0 mm
/run/beamOn 1000000
# run 8
/Fibra/setDistance 9.0 mm
/run/beamOn 1000000
# run 9
/Fibra/setDistance 1.0 cm
/run/beamOn 1000000
# run 10
/Fibra/setDistance 1.5 cm
/run/beamOn 1000000
# run 11
/Fibra/setDistance 2.0 cm
/run/beamOn 1000000
# run 12
/Fibra/setDistance 2.5 cm
/run/beamOn 1000000
```

Codice 39 - Macro file Co60-ang.mac per impostare i parametri delle simulazioni relativamente alla distribuzione isotropica nell'angolo solido del detector.

6.3 Risultati

Per ognuno dei due *script* lanciati il programma genera 13 *file* di testo ("`spettroID.txt`", dove `ID` è l'identificativo del *run* al quale il *file* fa riferimento) ciascuno contenente lo spettro dell'energia rilasciata nel rivelatore, e un *file* (`output.txt`) contenente un resoconto delle 13 simulazioni che riporta i conteggi degli *hit*, la configurazione del *run* (sorgente, distanza, distribuzione) e l'energia totale rilasciata nel rivelatore.

6.3.1 Risultati relativi a simulazioni con distribuzione isotropica in 4π

I risultati delle simulazioni con distribuzione isotropica in 4π sono riportati di seguito:

```
################ Run 0
            > Source Type : Co60
            > Distribution Type : iso
            > Source Distance : 1 mm
            > Primaries Hit count : 311460
            > Total Energy Deposit : 343.073 MeV
##################################

################ Run 1
            > Source Type : Co60
            > Distribution Type : iso
            > Source Distance : 2 mm
            > Primaries Hit count : 150802
            > Total Energy Deposit : 176.464 MeV
##################################

################ Run 2
            > Source Type : Co60
            > Distribution Type : iso
            > Source Distance : 3 mm
            > Primaries Hit count : 100621
            > Total Energy Deposit : 121.505 MeV
##################################

################ Run 3
            > Source Type : Co60
            > Distribution Type : iso
            > Source Distance : 4 mm
            > Primaries Hit count : 74857
            > Total Energy Deposit : 90.6768 MeV
##################################

################ Run 4
            > Source Type : Co60
            > Distribution Type : iso
            > Source Distance : 5 mm
            > Primaries Hit count : 60108
            > Total Energy Deposit : 79.6051 MeV
##################################

################ Run 5
            > Source Type : Co60
            > Distribution Type : iso
            > Source Distance : 6 mm
```

```
            > Primaries Hit count : 49584
            > Total Energy Deposit : 59.7926 MeV
   ##################################

   ################ Run 6
            > Source Type : Co60
            > Distribution Type : iso
            > Source Distance : 7 mm
            > Primaries Hit count : 42578
            > Total Energy Deposit : 49.6877 MeV
   ##################################

   ################ Run 7
            > Source Type : Co60
            > Distribution Type : iso
            > Source Distance : 8 mm
            > Primaries Hit count : 37322
            > Total Energy Deposit : 43.1183 MeV
   ##################################

   ################ Run 8
            > Source Type : Co60
            > Distribution Type : iso
            > Source Distance : 9 mm
            > Primaries Hit count : 33129
            > Total Energy Deposit : 38.7375 MeV
   ##################################

   ################ Run 9
            > Source Type : Co60
            > Distribution Type : iso
            > Source Distance : 1 cm
            > Primaries Hit count : 29882
            > Total Energy Deposit : 35.794 MeV
   ##################################

   ################ Run 10
            > Source Type : Co60
            > Distribution Type : iso
            > Source Distance : 1.5 cm
            > Primaries Hit count : 19822
            > Total Energy Deposit : 24.5788 MeV
   ##################################

   ################ Run 11
            > Source Type : Co60
            > Distribution Type : iso
            > Source Distance : 2 cm
            > Primaries Hit count : 14994
            > Total Energy Deposit : 19.6173 MeV
   ##################################

   ################ Run 12
            > Source Type : Co60
            > Distribution Type : iso
            > Source Distance : 2.5 cm
            > Primaries Hit count : 11968
            > Total Energy Deposit : 11.3311 MeV
   ##################################
```

Codice 40 – Resoconto delle simulazioni eseguite con distribuzione isotropica in 4π.

Tali valori sono stati riportati in forma tabellare ed elaborati per ottenere l'efficienza geometrica della fibra. In particolare ciascun conteggio è stato rapportato al numero di particelle generate. Il numero di particelle per evento nel caso della sorgente di ^{60}Co corrisponde al doppio degli eventi generati, in quanto tale

sorgente emette due raggi gamma per ciascun evento. Per cui ogni conteggio è stato rapportato al valore 2.000.000 ottenendo i valori riportati nella tabella seguente:

EFFICIENZA GEOMETRICA in funzione della distanza			
Source Distance (mm)	Efficienza geometrica	Efficienza teorica	Incertezza (± Δ)
1	0,15573	0,149605647	0,000279043
2	0,075401	0,074802823	0,000194166
3	0,0503105	0,049868549	0,000158604
4	0,0374285	0,037401412	0,0001368
5	0,030054	0,029921129	0,000122585
6	0,024792	0,024934274	0,000111337
7	0,021289	0,021372235	0,000103172
8	0,018661	0,018700706	9,65945E-05
9	0,0165645	0,01662285	9,10069E-05
10	0,014941	0,014960565	8,64321E-05
15	0,009911	0,00997371	7,03953E-05
20	0,007497	0,007480282	6,1225E-05
25	0,005984	0,005984226	5,46992E-05

Tabella 1 – Valori sperimentali e teorici dell'efficienza geometrica al variare della distanza.

Per quanto riguarda l'incertezza associata al valore dell'efficienza calcolato, essa è stata ricavata come:

$$\Delta = \pm \frac{\sqrt{N_H}}{N_{TOT}}$$

Tenendo conto che, per la legge dei grandi numeri, l'errore statistico commesso sulla stima N_H viene assunto come la radice quadrata del numero (e pertanto può essere reso piccolo aumentando il numero di eventi). Tali valori sono riportati in forma grafica nel Grafico 1:

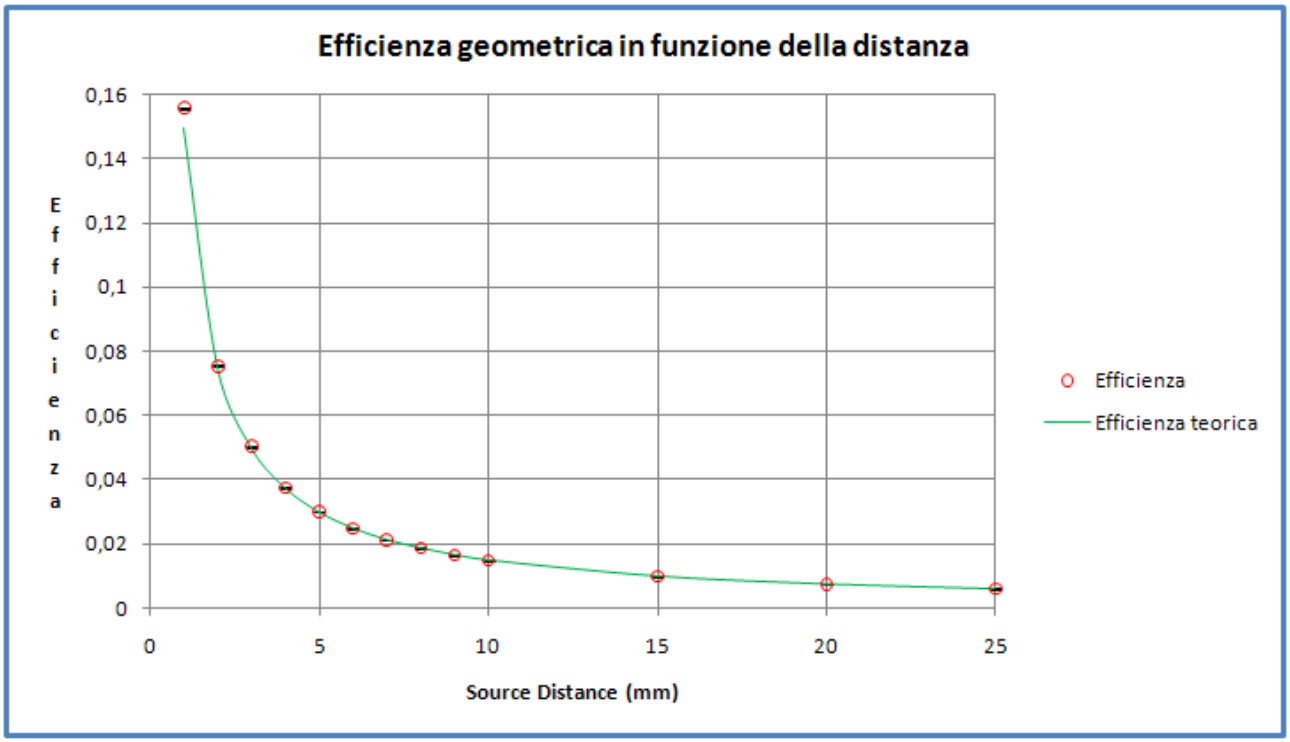


Grafico 1 – Efficienza geometrica del rivelatore in funzione della distanza.

Per ogni *run* è stato tracciato lo spettro dell'energia rilasciata nella fibra; prendiamo in considerazione ad esempio i due spettri, ancora una volta nel caso di simulazioni con distribuzione isotropica in 4π, relativi ai *run* eseguiti a distanza rispettivamente 1 mm e 1 cm (Grafico 2 e Grafico 3).

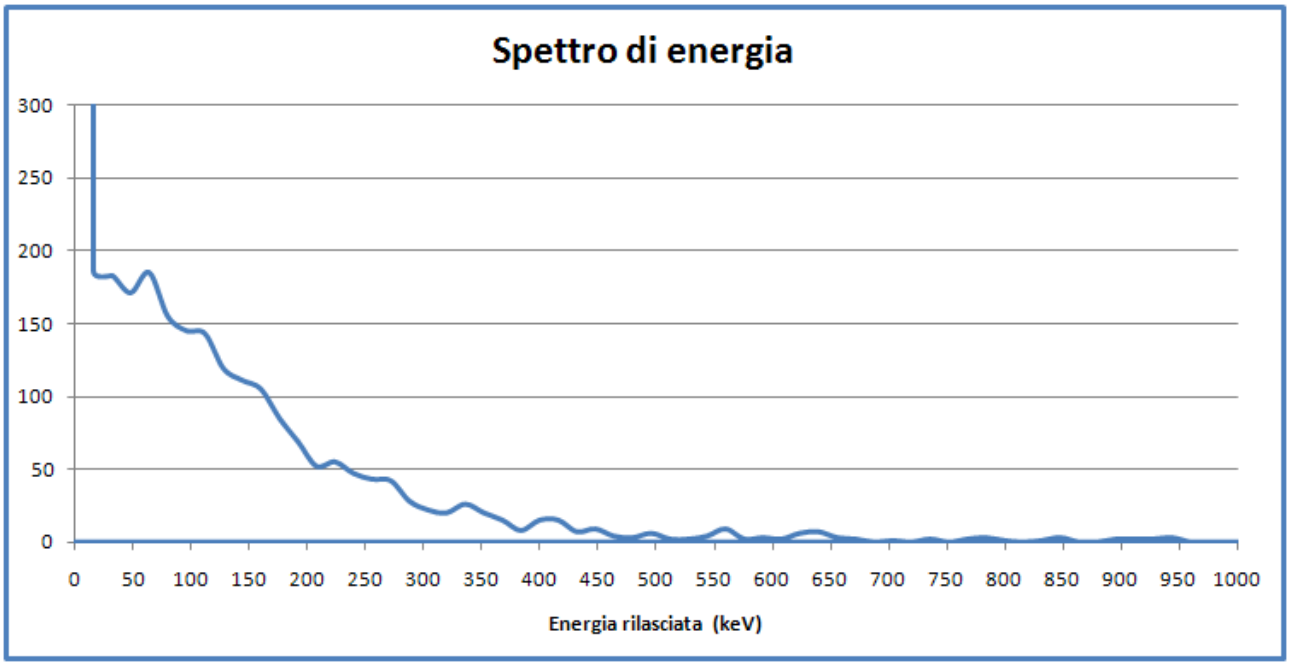


Grafico 2 – Spettro di energia relativo alla simulazione con distribuzione dei raggi isotropica in 4π nel caso di sorgente posta a 1 mm dalla fibra.

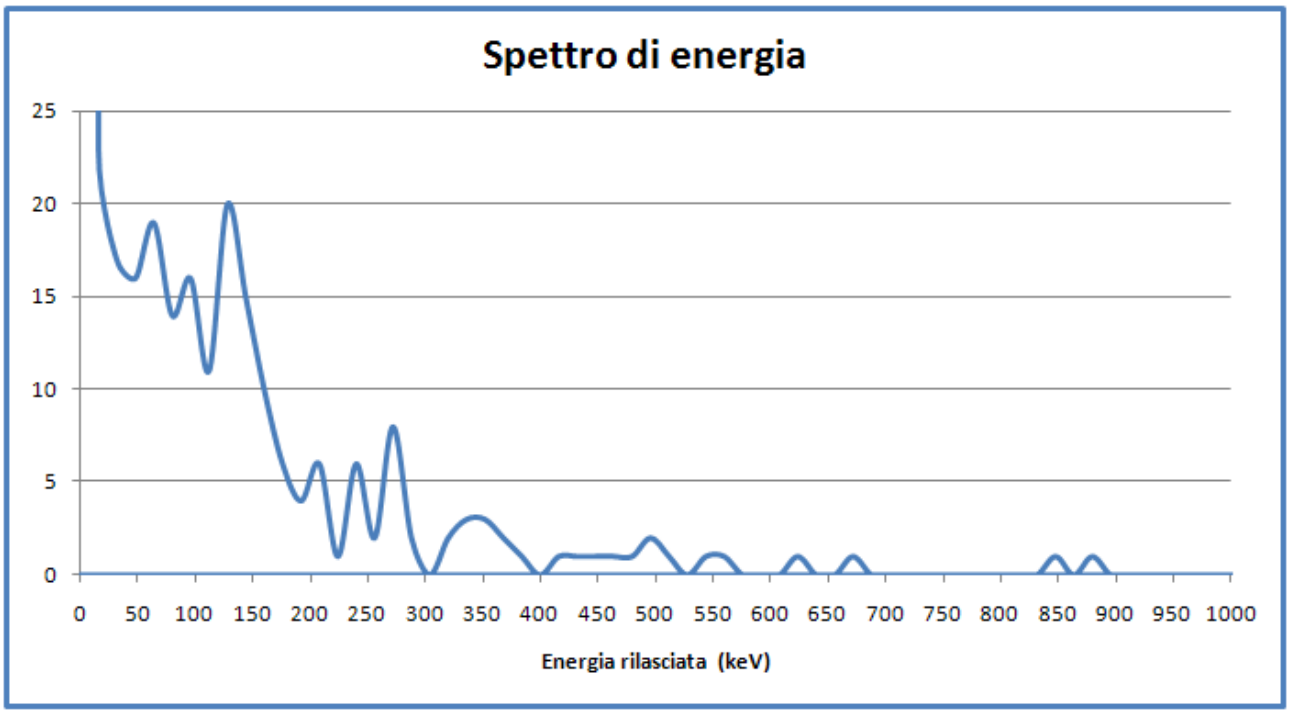


Grafico 3 - Spettro di energia relativo alla simulazione con distribuzione dei raggi isotropica in 4π nel caso di sorgente posta a 1 cm dalla fibra.

Questi risultati risentono pesantemente della scarsa statistica come facilmente visibile dall'entità delle fluttuazioni. Allo scopo di ridurre tali fluttuazioni è auspicabile disporre di una migliore statistica, come verrà illustrato nel paragrafo seguente.

6.3.2 Risultati relativi a simulazioni con distribuzione isotropica nell'angolo solido

Successivamente sono state svolte simulazioni generando gli eventi emettendo selettivamente particelle gamma nell'angolo solido sotteso dalla fibra, allo scopo di ottenere risultati statisticamente più significativi risparmiando nel tempo di esecuzione ed evitando di elaborare tutti quegli eventi in cui le particelle non colpiscono il *detector*.

In questo caso 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 è riportato il file di *output* con il resoconto delle simulazioni:

```
################ Run 0
            > Source Type : Co60
            > Distribution Type : solidAngle
            > Source Distance : 1 mm
            > Primaries Hit count : 1999881
            > Total Energy Deposit : 2.36624 GeV
##################################

################ Run 1
            > Source Type : Co60
            > Distribution Type : solidAngle
            > Source Distance : 2 mm
            > Primaries Hit count : 1936931
            > Total Energy Deposit : 2.21315 GeV
##################################

################ Run 2
            > Source Type : Co60
            > Distribution Type : solidAngle
            > Source Distance : 3 mm
            > Primaries Hit count : 1905235
            > Total Energy Deposit : 2.19941 GeV
##################################

################ Run 3
            > Source Type : Co60
            > Distribution Type : solidAngle
            > Source Distance : 4 mm
            > Primaries Hit count : 1894192
            > Total Energy Deposit : 2.24822 GeV
##################################

################ Run 4
            > Source Type : Co60
            > Distribution Type : solidAngle
            > Source Distance : 5 mm
```

```
            > Primaries Hit count : 1888788
            > Total Energy Deposit : 2.22665 GeV
   ###################################

   ################ Run 5
            > Source Type : Co60
            > Distribution Type : solidAngle
            > Source Distance : 6 mm
            > Primaries Hit count : 1885922
            > Total Energy Deposit : 2.12263 GeV
   ###################################

   ################ Run 6
            > Source Type : Co60
            > Distribution Type : solidAngle
            > Source Distance : 7 mm
            > Primaries Hit count : 1884292
            > Total Energy Deposit : 2.19444 GeV
   ###################################

   ################ Run 7
            > Source Type : Co60
            > Distribution Type : solidAngle
            > Source Distance : 8 mm
            > Primaries Hit count : 1883488
            > Total Energy Deposit : 2.18781 GeV
   ###################################

   ################ Run 8
            > Source Type : Co60
            > Distribution Type : solidAngle
            > Source Distance : 9 mm
            > Primaries Hit count : 1883131
            > Total Energy Deposit : 2.18693 GeV
   ###################################

   ################ Run 9
            > Source Type : Co60
            > Distribution Type : solidAngle
            > Source Distance : 1 cm
            > Primaries Hit count : 1881705
            > Total Energy Deposit : 2.16475 GeV
   ###################################

   ################ Run 10
            > Source Type : Co60
            > Distribution Type : solidAngle
            > Source Distance : 1.5 cm
            > Primaries Hit count : 1880754
            > Total Energy Deposit : 2.16694 GeV
   ###################################

   ################ Run 11
            > Source Type : Co60
            > Distribution Type : solidAngle
            > Source Distance : 2 cm
            > Primaries Hit count : 1881021
            > Total Energy Deposit : 2.17129 GeV
   ###################################

   ################ Run 12
            > Source Type : Co60
            > Distribution Type : solidAngle
            > Source Distance : 2.5 cm
            > Primaries Hit count : 1880608
            > Total Energy Deposit : 2.09616 GeV
   ###################################
```

Codice 41 - Resoconto delle simulazioni eseguite con distribuzione isotropica nell'angolo solido della fibra.

Inoltre è stata valutata l'efficienza di rivelazione in funzione di alcuni valori di soglia di energia stabiliti, che sarà riportata in tabella e graficamente nel seguito.

In virtù della miglior statistica gli spettri di energia ottenuti nel caso di distribuzione isotropica nell'angolo solido 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 rispettivamente ad 1 mm e 1 cm dal *detector*.

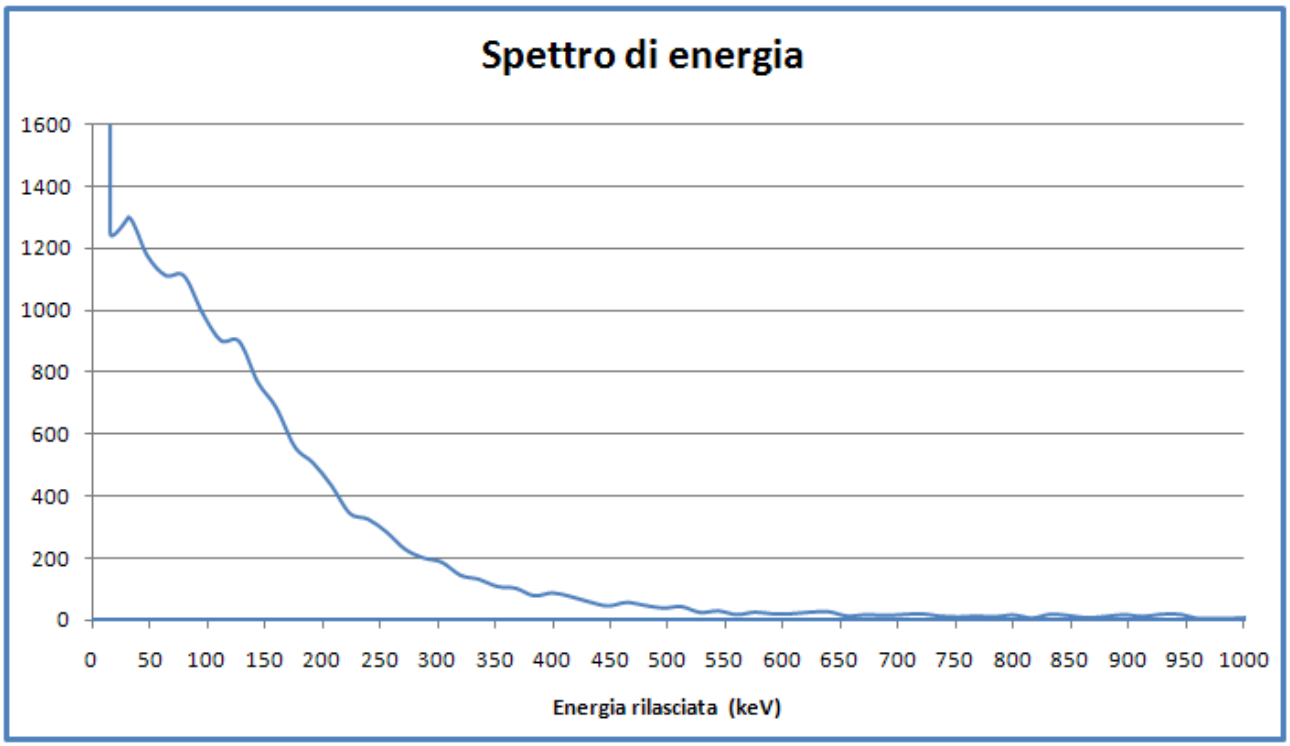


Grafico 4 - Spettro di energia relativo alla simulazione con distribuzione dei raggi isotropica nell'angolo solido della fibra nel caso di sorgente posta a 1 mm dalla fibra.

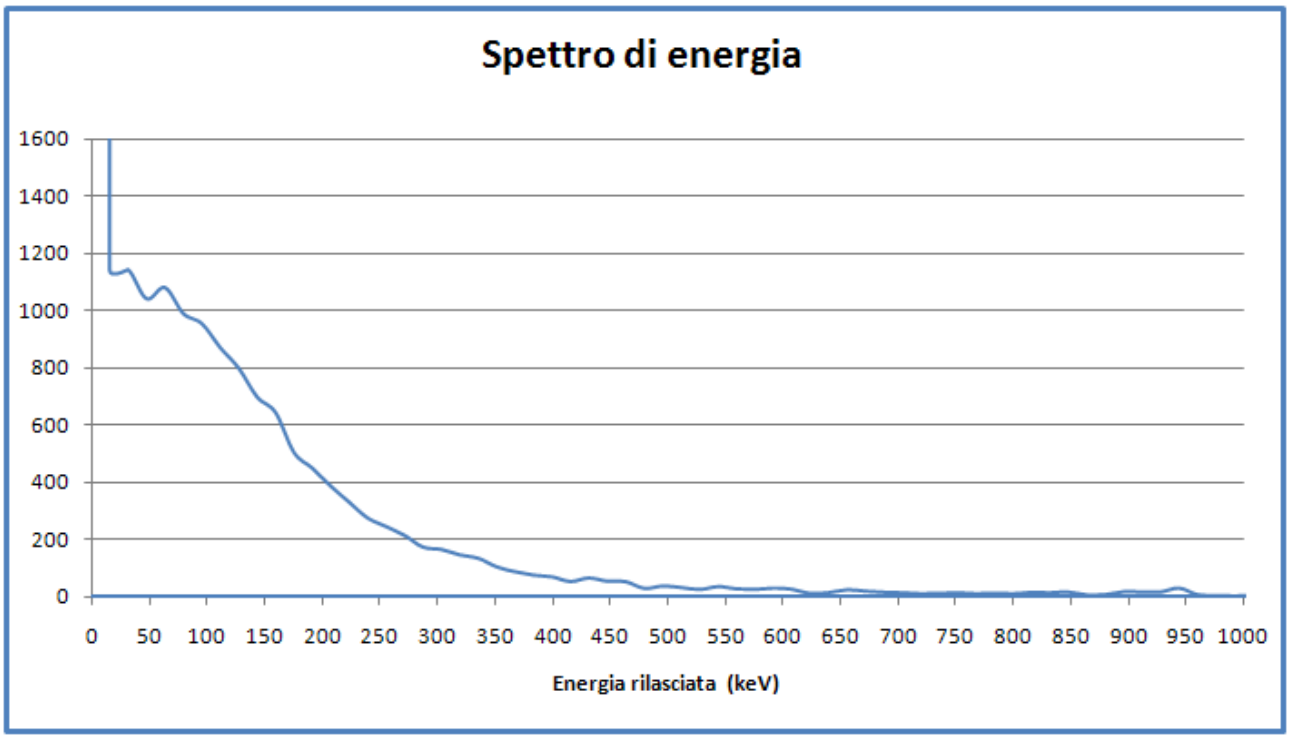


Grafico 5 - Spettro di energia relativo alla simulazione con distribuzione dei raggi isotropica nell'angolo solido della fibra nel caso di sorgente posta a 1 cm dalla fibra.

Gli spettri di energia sopra riportati hanno permesso di valutare l'efficienza di rivelazione che è stata ottenuta caso per caso e valutata sulla base di 10 valori di soglie di energia differenti, ottenendo i risultati riportati di seguito.

Un primo esempio è costituito dall'efficienza di rivelazione ottenuta nel caso di simulazione con sorgente posta ad 1mm di distanza dalla fibra, riportata nella seguente tabella (Tabella 2):

EFFICIENZA DI RIVELAZIONE			
THRESHOLD (keV)	Σ hit (e>th)	ε(th)	$\pm \Delta_{th}$
10	15096	0,00117545	0,000304895
20	14305	0,001113859	0,000303595
30	13534	0,001053825	0,000302323
40	12725	0,000990832	0,000300982
50	11929	0,000928852	0,000299656
100	8448	0,000657804	0,000293791
150	5652	0,000440093	0,000288993
200	3671	0,000285842	0,000285545

Tabella 2 – Efficienza di rivelazione nel caso di sorgente posta ad 1 mm dalla fibra.

In questo caso l'incertezza associata all'efficienza di rivelazione è stata ricavata con la formula:

$$\Delta_{th} = \sqrt{\left(\frac{N_{Hth}}{N_{TOT}}\right)^2 + \Delta^2}$$

Dove N_{Hth} si riferisce al numero di particelle incidenti che hanno rilasciato energia maggiore della soglia; N_{TOT} è il numero di eventi generati; Δ è l'incertezza associata all'efficienza geometrica.

Gli stessi valori sono stati calcolati per tutti i 13 *run* eseguiti che sono stati raggruppati in un'unica tabella (Tabella 3) e in un unico grafico (Grafico 6), come mostrato di seguito:

EFFICIENZA DI RIVELAZIONE in funzione della distanza								
THRESHOLD (keV)	10 keV	20 keV	30 keV	40 keV	50 keV	100 keV	150 keV	200 keV
SOURCE DISTANCE (mm)	$\varepsilon_{(th)}$	$\varepsilon_{(th)}$	$\varepsilon_{(th)}$	$\varepsilon_{(th)}$	$\varepsilon_{(th)}$	$\varepsilon_{(th)}$	$\varepsilon_{(th)}$	$\varepsilon_{(th)}$
1	0,00117545	0,001113859	0,001053825	0,000990832	0,000928852	0,000657804	0,000440093	0,000285842
2	0,000535196	0,000505752	0,000476949	0,000448598	0,000420059	0,000294629	0,000198795	0,000128596
3	0,000352677	0,000333634	0,000315648	0,000295826	0,00027862	0,000197192	0,000130178	8,42701E-05
4	0,000265742	0,00025107	0,000236455	0,000222756	0,000209019	0,000148535	9,92978E-05	6,64169E-05
5	0,000212827	0,000201076	0,000190077	0,000179558	0,000167957	0,00011963	7,87114E-05	5,15877E-05
6	0,000170296	0,00016095	0,000151368	0,000142219	0,000133666	9,44079E-05	6,26866E-05	4,09192E-05
7	0,000148821	0,000140805	0,000132769	0,000124743	0,000117015	8,28461E-05	5,46914E-05	3,58081E-05
8	0,00013003	0,000123163	0,000116137	0,000109241	0,000102542	7,2526E-05	4,84999E-05	3,16864E-05
9	0,000115695	0,000108986	0,000102733	9,68526E-05	9,10633E-05	6,42785E-05	4,24879E-05	2,75799E-05
10	0,000103437	9,75797E-05	9,2425E-05	8,68147E-05	8,19589E-05	5,7747E-05	3,81145E-05	2,48618E-05
15	6,80886E-05	6,42976E-05	6,06504E-05	5,70576E-05	5,38217E-05	3,83754E-05	2,52285E-05	1,67793E-05
20	5,19167E-05	4,89891E-05	4,62415E-05	4,37075E-05	4,12747E-05	2,9392E-05	1,92823E-05	1,243E-05
25	4,02753E-05	3,81689E-05	3,60386E-05	3,39113E-05	3,18887E-05	2,23652E-05	1,46129E-05	9,60133E-06

Tabella 3 – Efficienza di rivelazione in funzione della distanza.

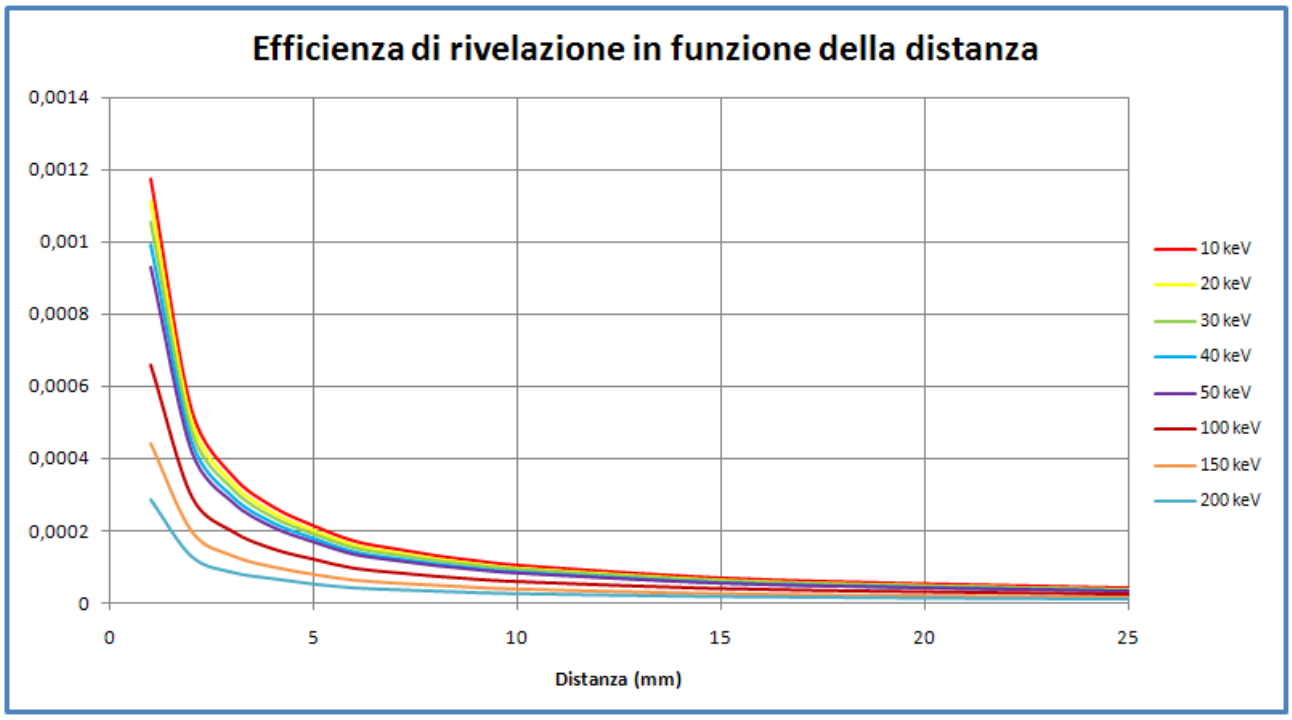


Grafico 6 – Efficienza di rivelazione in funzione della distanza.

Gli stessi dati possono essere espressi in funzione della soglia, come nel grafico seguente:

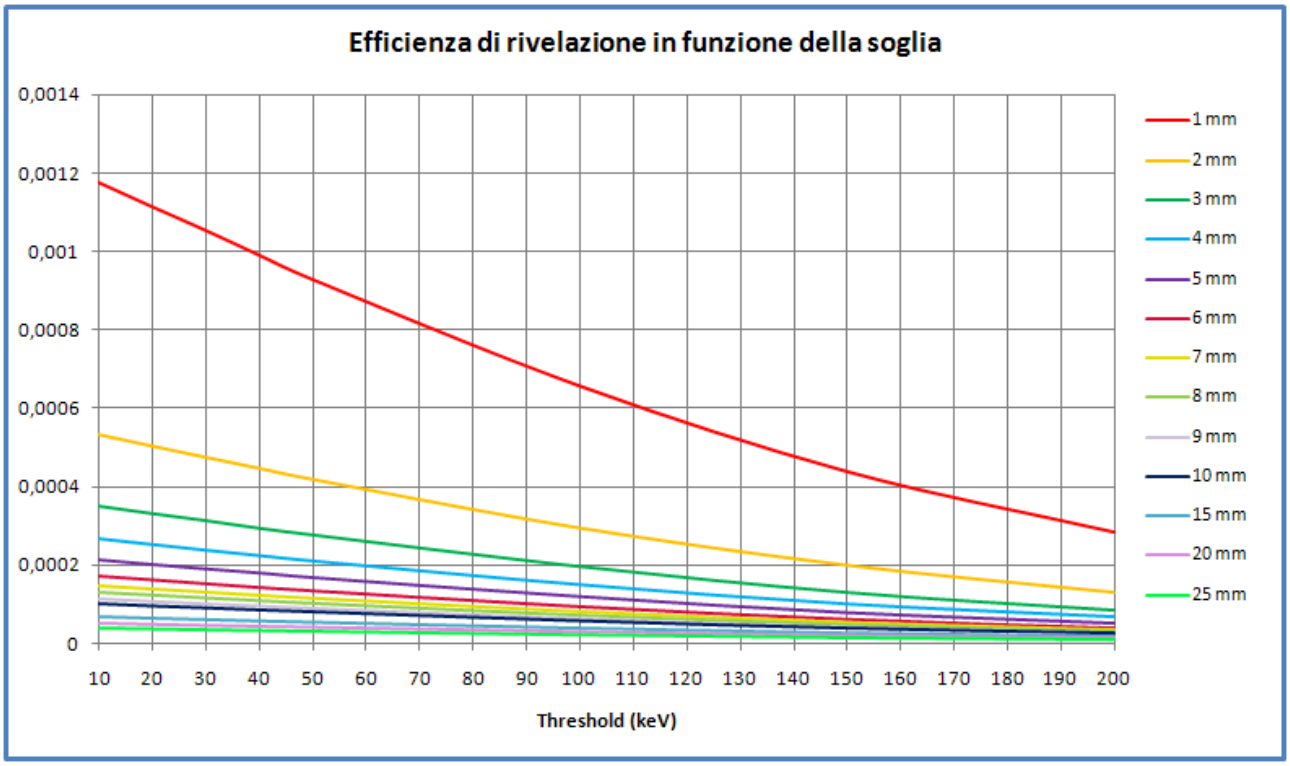


Grafico 7 – Efficienza di rivelazione in funzione della soglia.

Il grafico seguente è stato ottenuto interpolando i dati mancanti:

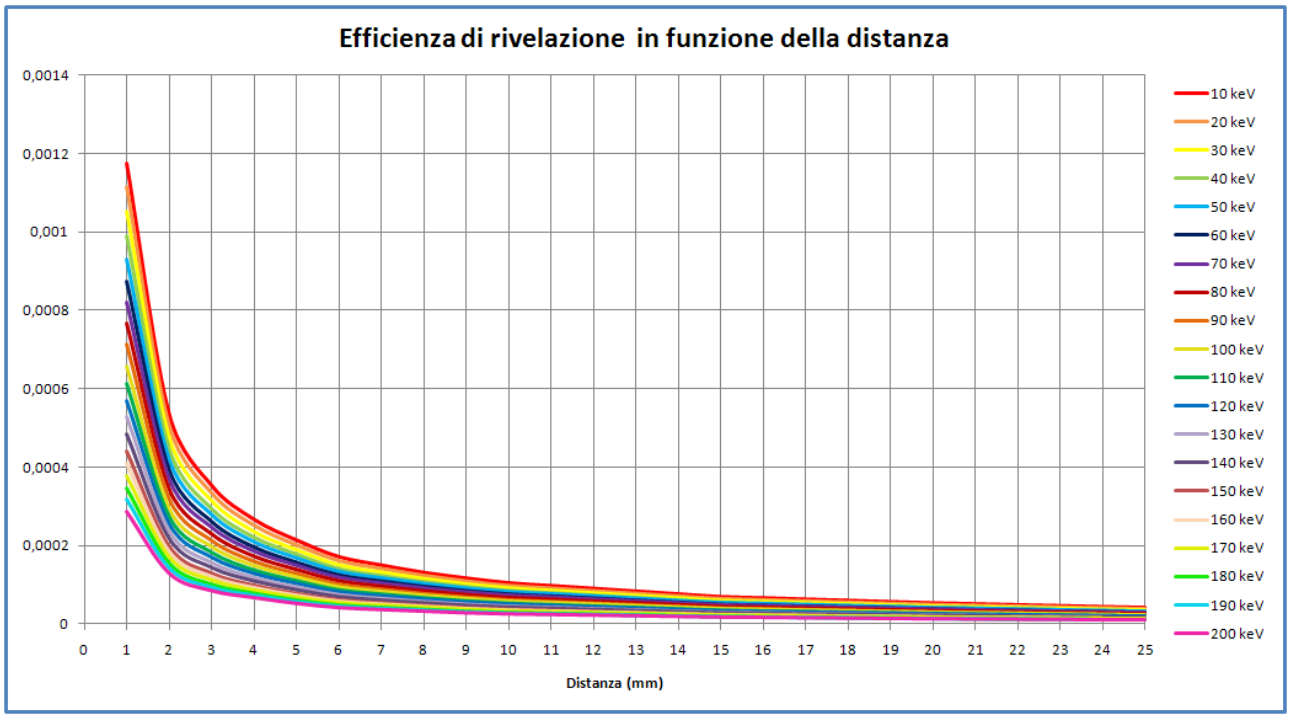
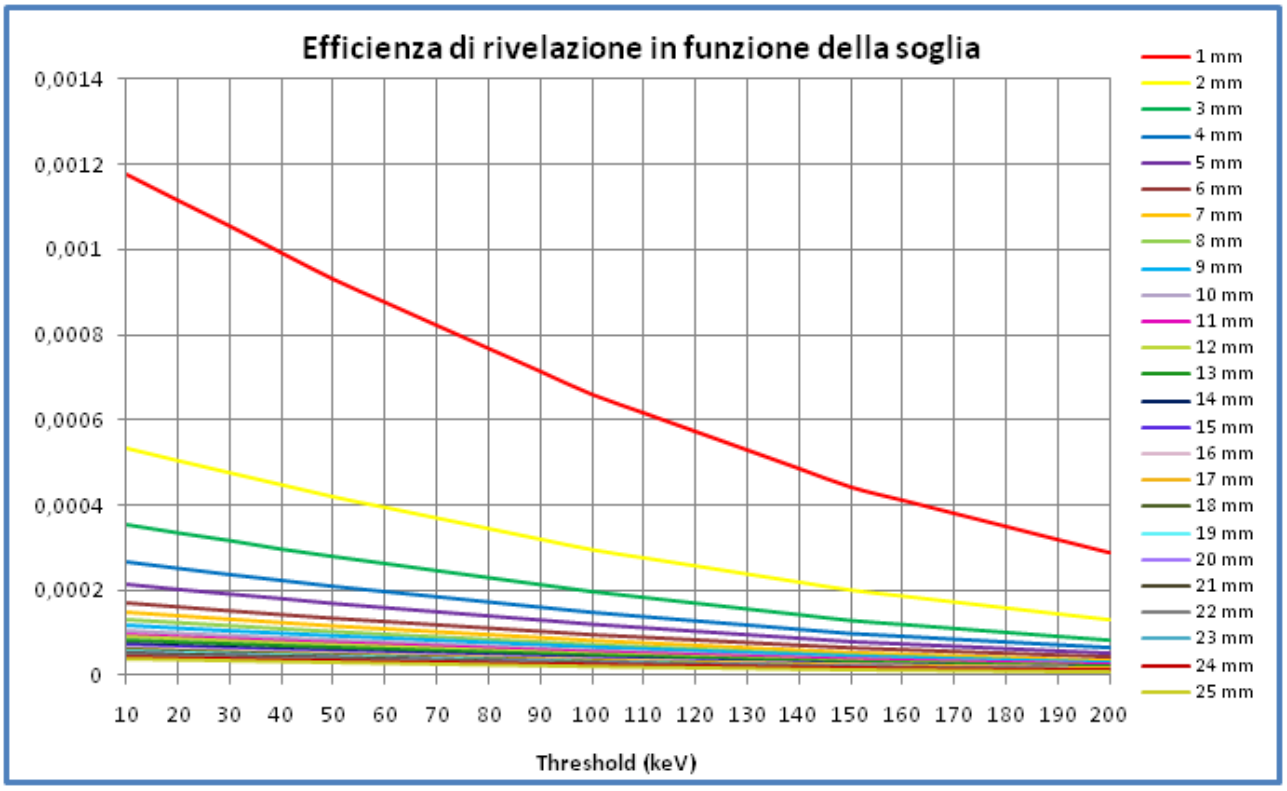


Grafico 8 – Efficienza di rivelazione in funzione della distanza (valori interpolati).



Il grafico seguente evidenzia l'andamento dell'efficienza di rivelazione in funzione sia della distanza che della soglia:

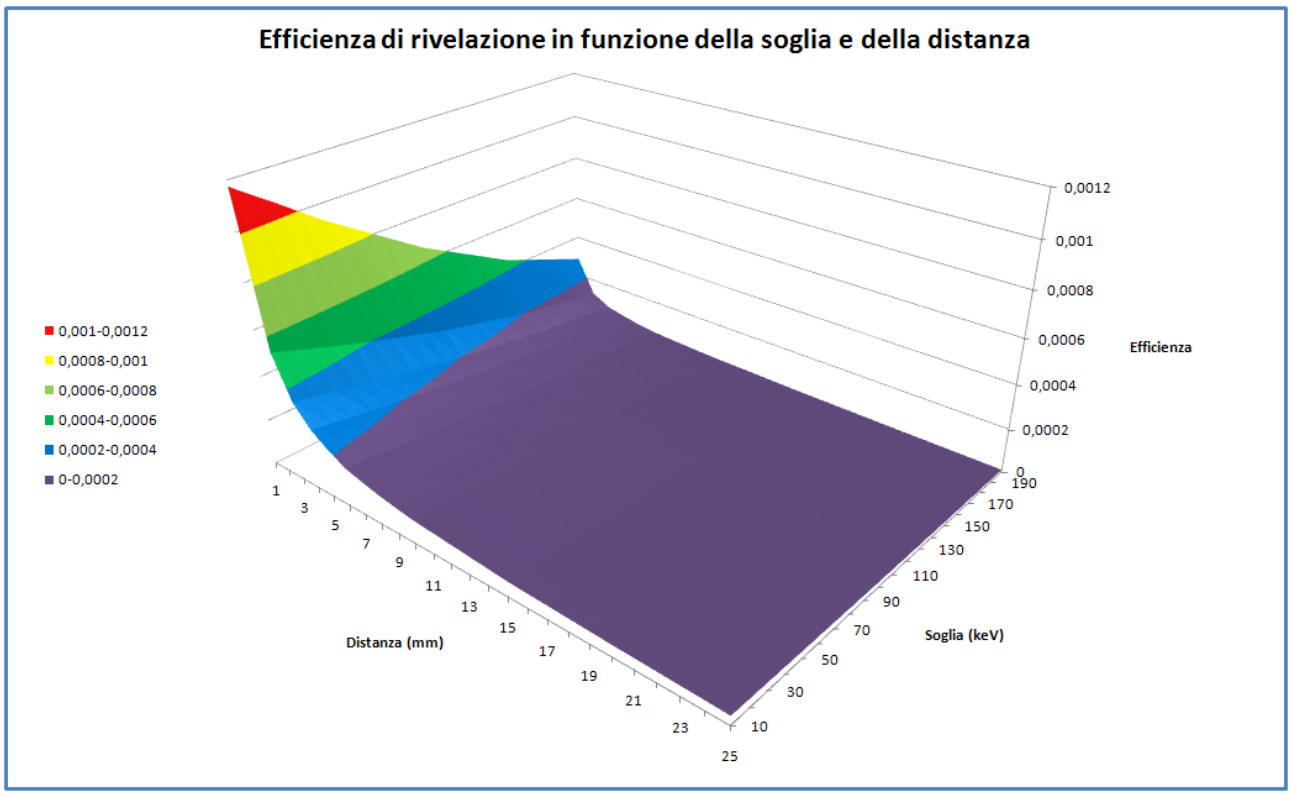


Grafico 9 – Efficienza di rivelazione in funzione della soglia e della distanza.

Conclusioni

Nella presente tesi è stato discusso il lavoro svolto presso i Laboratori Nazionali del Sud dell'INFN per lo sviluppo di un'applicazione per il *toolkit* di simulazione Geant4 con l'obiettivo di simulare una fibra scintillante di polistirene soggetta all'emissione di raggi gamma da parte di una sorgente radioattiva di cobalto-60 al fine di valutarne la risposta in termini di efficienza geometrica, spettro energetico ed efficienza di rilevazione.

Tale valutazione condotta per mezzo del simulatore Geant4, ampiamente descritto nel capitolo 3, si inquadra nel contesto della problematica del monitoraggio delle scorie radioattive che attualmente vede nascere presso i LNS dell'Istituto Nazionale di Fisica Nucleare il progetto DMNR per il monitoraggio *real-time* dei fusti di rifiuti radioattivi stoccati nei depositi delle centrali nucleari. Tale progetto prevede l'impiego delle fibre scintillanti simulate per costituire una matrice di rilevazione attorno ai fusti.

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, fino alla simulazione dell'intera matrice di rivelazione, inglobando nel processo di simulazione le interazioni ottiche che intervengono ai fini della rilevazione per mezzo dei fotosensori. In tal modo sarebbe possibile simulare ad esempio particolari situazioni anomale, come una perdita da un fusto dovuta a danneggiamenti accidentali, etc. e valutare di conseguenza la risposta del sistema con la possibilità di determinare l'entità e l'effetto dei danni che tali incidenti potrebbero causare, studiando delle possibili soluzioni.

Ringraziamenti

Ringrazio il prof. Malgeri per avermi dato il suo supporto rivolgendomi grande disponibilità e cortesia durante l'attività di tirocinio e la stesura di questa tesi.

I miei ringraziamenti più sentiti li rivolgo a mio zio Paolo per la sua pazienza, per la disponibilità e per avermi guidato durante l'attività di tirocinio con le sue conoscenze e la sua simpatia.

Un grazie particolare per mia zia Raffaela che, con grande affetto e immensa disponibilità, mi ha accompagnato durante la stesura di questa tesi con i suoi preziosi consigli (talvolta combattendo insieme a me con le insidie di Microsoft Word), ma soprattutto trasmettendomi l'energia per portarla a termine.

Colgo l'occasione per ringraziare la mia famiglia e gli amici che hanno condiviso con me le gioie, ma anche i momenti difficili, durante il percorso che mi ha condotto a questo importante traguardo. A voi un altrettanto immenso Grazie per avermi supportata (a volte sopportata :-)) e per avermi spronato sempre al meglio.

Vale

Bibliografia

[1] Finocchiaro, P., *DMNR: a new concept for real-time online monitoring of short and medium term radioactive waste*, in corso di pubblicazione

[2] Bellini, V., et al.; *Implementation of a new low cost detector for low intensity light pulses produced by a radioactive gamma-source*, Applied Radiation and Isotopes, 68, (2010) 1320–1323

[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, pagg 77-100

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

[5] Metropolis, N., *The beginning of the Monte Carlo Method*, Los Alamos Science Special Issue, (1987), pagg. 125-130

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

[7] Agostinelli, S., et al.: *Geant4 - a simulation toolkit*, Nuclear Instruments and Methods in Physics Research A 506 (2003), pagg. 250-303

[8] L'Annunziata, Michael F.. *Radioactivity: introduction and history*. Amsterdam, Netherlands: Elsevier BV. (2007) ISBN 9780444527158

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

[10] "*Rayleigh scattering*" su HyperPhysics

(http://hyperphysics.phy-astr.gsu.edu/hbase/atmos/blusky.html#c2)

[11] Eberhard Haug & Werner Nakel, *The elementary process of bremsstrahlung*, River Edge NJ: World Scientific. p. Scientific lecture notes in physics, vol. 73, (2004), ISBN 981-238-578-9

[12] 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)

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

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

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

[16] Geant4 Physics reference Manual (*http://geant4.cern.ch/G4UsersDocuments/UsersGuides/PhysicsReferenceManual/html/PhysicsReferenceManual.html*)

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

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

[19] Weisstein, Eric W., "*Solid Angle*" from MathWorld - A Wolfram Web Resource (http://mathworld.wolfram.com/SolidAngle.html)

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

[21] 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

[22] Grady Booch, *Object-oriented Analysis and Design with Applications* (2nd ed.), Benjamin Cummings, Redwood City, (1993), ISBN 0-8053-5340-2