UNIVERSITÀ DEGLI STUDI DI CATANIA

Facoltà di Ingegneria

Corso di Laurea Magistrale Ingegneria Informatica

TESI DI LAUREA

MONITORAGGIO DI RIFIUTI RADIOATTIVI: SIMULAZIONE DI SORGENTI ESTESE IN GEOMETRIE REALISTICHE.

Laura Curcuruto

Relatore:

Prof. Michele Malgeri

Correlatore:

Dott. Paolo Finocchiaro

ANNO ACCADEMICO 2011/2012

Indice

pag.

Introduzione

Il presente lavoro di tesi è stato svolto presso i Laboratori Nazionali del Sud dell'INFN[1] ed è collegato al progetto DMNR (Detector Mesh of Nuclear Repositories) che è attualmente in fase di sviluppo presso INFN-LNS di Catania e nasce da una collaborazione con Ansaldo[2]. Esso prevede la realizzazione di un prototipo fatto da una rete di rivelatori per il monitoraggio online di siti di deposito di scorie radioattive a medio e breve termine.

L'obiettivo di questo lavoro è quello di realizzare un modello di simulazione che permetta uno studio qualitativo e quantitativo del problema del monitoraggio e della rivelazione della radiazione gamma proveniente dai rifiuti radioattivi che fuoriescono dai fusti di contenimento. Nello specifico riguarda lo sviluppo di un'applicazione che permette di simulare la rivelazione della radiazione gamma prodotta da sorgenti radioattive estese attraverso l'uso di geometrie di fibre scintillanti di polistirene.

Le simulazioni sono state eseguite realizzando una distribuzione di tipo isotropica in tutto lo spazio, ottenuta con una sorgente estesa, ed in particolare sono state simulate tre diverse geometrie:

- sorgente estesa con tre fusti;
- sorgente estesa con sei fusti;
- sorgente estesa con nove fusti.

Oltre a variare la geometria, sono stati variati altri parametri come la posizione della sorgente , il numero di particelle emesse, etc... con lo scopo di valutare il numero di particelle incidenti per ricavare l'efficienza geometrica della fibra.

L'applicazione è stata realizzata utilizzando il toolkit Geant4, che permette di simulare tutte le particelle conosciute all'interno di una qualsivoglia geometria e materiale usando gli algoritmi Monte Carlo[4].

---

[1] L'Istituto Nazionale di Fisica Nucleare è l'ente italiano che promuove, coordina ed effettua la ricerca scientifica nel campo della fisica nucleare, subnucleare e astro particellare, nonché lo sviluppo tecnologico necessario alle attività in tali settori.

[2] Ansaldo Nucleare s.p.a. è un'azienda italiana che opera nel settore nucleare, realizzando centrali nucleari di terza generazione.

Geant4, scritto in linguaggio di programmazione C++ e con la tecnica Object Oriented, presenta una notevole versatilità di utilizzo, dando la possibilità di suddividere lavori, anche molto complessi, in piccoli blocchi più semplici comunicanti tra loro. Tale toolkit è sviluppato da un centinaio di ricercatori sparsi in tutto il mondo ed è utilizzato per la progettazione e il test degli esperimenti al CERN, ma sta traendo anche notevole diffusione in altri ambienti come la fisica dello spazio e la fisica medica.

L'applicazione è stata sviluppata ed eseguita su un processore Intel Core i3, Sistema Operativo Linux Ubuntu 12.04 LTS utilizzando la versione del toolkit di Geant 4.9.p02, anche se attualmente è disponibile una versione aggiornata.

In particolare in questa tesi l'attenzione verrà focalizzata sui seguenti argomenti:

- DMNR
- Metodo Monte Carlo
- Geant4
- Fibra scintillante
- Sviluppo dell'applicazione
- Simulazioni
- Analisi dei risultati

1. DMNR

DMNR è l'acronimo di *Detector Mesh for Nuclear Repositories* ed è un progetto, attualmente in fase di sviluppo presso INFN-LNS Catania[1], che mira alla realizzazione di un prototipo capace di implementare il monitoraggio on-line di breve-medio termine dei rifiuti radioattivi. Tale progetto è stato testato attraverso un prototipo presso la ex centrale nucleare di Garigliano, situata nel comune di Sessa Aurunca in provincia di Caserta, fornendo risultati soddisfacenti.

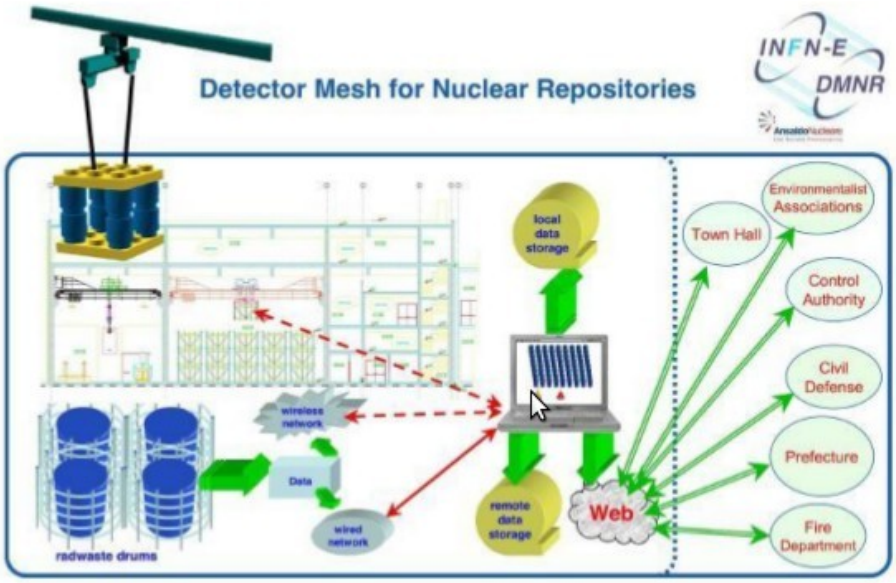


Fig. 1: Prototipo Detector Mesh for Nuclear Repositories

Il prototipo è caratterizzato da una fitta maglia di sensori, si tratta di fibre scintillanti lette alle estremità tramite Silicon PhotoMultipliers (SiPM), da installare sui bidoni contenenti i rifiuti nucleari. Il flusso di dati provenienti dai sensori verso una console è gestito attraverso un'elettronica di front-end e un sistema di conteggio basati su FPGA[3] (Field Programmable Gate Array), che si occupa della trasmissione dei dati ridondanti, di una interfaccia utente grafica e un sistema di archiviazione dati. Inoltre è possibile effettuare un' ispezione remota ad hoc attraverso un braccio

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

.

robotico, dotato sia di una telecamera per l' ispezione visiva che di rivelatori di radiazione ad alta risoluzione.

1.1. Rifiuti radioattivi e decadimento radioattivo

Molte attività umane producono rifiuti radioattivi: la diagnostica e la terapia medica (per es. la radioimmunologia, la radioterapia), la ricerca scientifica, l'industria agroalimentare (per es. la sterilizzazione delle derrate per irraggiamento), i controlli di produzione industriale (per es. le radiografie di saldature). Questi rifiuti, per un tempo variabile da pochi istanti a milioni di anni, emettono radiazioni che possono avere effetti negativi sull'ambiente e sull'uomo, ma che sono comunque di intensità decrescente nel tempo, per il fenomeno del "decadimento radioattivo".

In Italia, i rifiuti radioattivi sono classificati in tre categorie, secondo il grado di pericolosità radiologica:

- *I Categoria:* rifiuti radioattivi la cui radioattività decade fino al livello del fondo naturale in tempi dell'ordine di mesi o al massimo di qualche anno. A questa categoria appartengono una parte dei rifiuti da impieghi medici o di ricerca scientifica;
- *II Categoria:* rifiuti radioattivi a bassa/media attività o a vita breve, che perdono quasi completamente la loro radioattività in un tempo dell'ordine di qualche secolo;
- *III Categoria:* rifiuti radioattivi ad alta attività o a vita lunga, per il decadimento dei quali sono necessari periodi molto più lunghi, da migliaia a centinaia di migliaia di anni.

I rifiuti radioattivi devono essere gestiti secondo criteri di sicurezza, che ne impediscano la dispersione nell'ambiente, impedendone il contatto con le catene alimentari per la generazione attuale e per quelle future. Ed è proprio per ridurre la loro pericolosità e assicurarne la gestione in sicurezza, che essi vengono sottoposti a:

- Condizionamento
- Smaltimento in depositi definitivi

Sono, inoltre, in corso in tutto il mondo attività di ricerca finalizzate alla riduzione con sistemi nucleari innovativi dei quantitativi e della tossicità dei rifiuti radioattivi. Si tratta, in particolare, dei nuovi s*istemi nucleari di IV generazione* (GEN IV) che permetteranno, in futuro, di "bruciare" i rifiuti radioattivi sfruttandone, nel contempo, il loro potenziale energetico.

I rifiuti di *I Categoria* vengono semplicemente immagazzinati in condizioni controllate fino a quando i valori di radioattività non ne permettono lo smaltimento. Poi vengono gestiti come rifiuti convenzionali o speciali.

I rifiuti di *II* e *III Categoria* vengono, invece, sottoposti al "condizionamento", cioè a trattamenti chimici e fisici che li convertono in forma solida, stabile e duratura adatta per la manipolazione, il trasporto e infine lo smaltimento in depositi dedicati. Il rifiuto condizionato è, dunque, un manufatto costituito dal materiale radioattivo inglobato in un materiale inerte, generalmente cemento o vetro, posto in un contenitore esterno costituito da un fusto in acciaio.

Lo smaltimento dei rifiuti radioattivi condizionati prevede la collocazione definitiva dei manufatti in un deposito, con l'intenzione di non recuperarli; il deposito deve garantire il completo isolamento dalla popolazione e dall'ambiente fino a quando la radioattività, per effetto del decadimento, non raggiunga valori paragonabili a quelli del fondo ambientale.
Nel caso dei *rifiuti radioattivi di bassa/media attività,* l'isolamento deve essere garantito per qualche secolo e quindi la soluzione di smaltimento più idonea è il *deposito superficiale*, protetto prevalentemente da barriere artificiali, cioè progettate e realizzate dall'uomo. I manufatti possono essere, ad esempio, immagazzinati all'interno di "moduli" prefabbricati in calcestruzzo armato, i quali sono poi stoccati in strutture scatolari, anch'esse in calcestruzzo armato, che vengono poi coperte ed interrate. Questo tipo di depositi è molto utilizzato e ne esistono ormai almeno un centinaio in tutto il mondo; i più moderni e avanzati si trovano in Francia, Spagna, Svezia, Giappone, Regno Unito, USA.

I *rifiuti radioattivi di III categoria*, ad alta attività o a lunga vita, mantengono invece livelli di radioattività significativi per decine e centinaia di migliaia di anni. Per l'isolamento di tali rifiuti non è possibile fare affidamento solo su barriere artificiali,

ma ci si deve affidare a barriere naturali. Vengono prese, quindi, in considerazione formazioni geologiche in profondità (600-800 metri e oltre) che presentino adeguate caratteristiche di stabilità e impermeabilità come, ad esempio, giacimenti di salgemma o formazioni argillose o di granito. L'unico esempio al mondo di deposito geologico operativo è il WIPP (Waste Isolation Pilot Plant), un deposito di smaltimento geologico in funzione negli USA dal marzo 1999, riservato ai rifiuti contenenti plutonio di produzione militare.

Per quanto riguarda il decadimento radioattivo possiamo dire che esso è un processo nel quale i nuclei degli atomi di sostanze radioattive si disintegrano rilasciando radiazioni seguendo un a propria legge (*legge del decadimento radioattivo*).
Esistono tre diversi tipi di decadimenti radioattivi, che si differenziano dal tipo di particella emessa a seguito del decadimento :

- *decadimento alfa (α):* consideriamo un nucleo con numero atomico Z e numero di massa A. In seguito ad un decadimento alfa, il nucleo emette una particella α , cioè un nucleo di elio composto da due protoni e due neutroni, e si trasforma in un nucleo diverso, con numero atomico (Z - 2) e numero di massa (A – 4). Un esempio è il decadimento dell'uranio-238 in torio-234. Le radiazioni alfa, per la loro natura, sono poco penetranti e possono essere completamente bloccate da un semplice foglio di carta.
- *decadimento beta (β):* il nucleo emette un elettrone e un antineutrino di tipo elettronico e si trasforma in un nucleo con numero atomico (Z + 1) ma stesso numero di massa A. Un esempio è il decadimento del Cobalto-60 in Nichel-60. Le radiazioni beta sono più penetranti di quelle alfa, ma possono essere completamente bloccate da piccoli spessori di materiali metallici (ad esempio, pochi millimetri di alluminio).
- *decadimento gamma (γ):* il nucleo non si trasforma ma passa semplicemente in uno stato di energia inferiore ed emette un fotone. La radiazione gamma accompagna solitamente una radiazione alfa o una radiazione beta. Infatti, dopo l'emissione alfa o beta, il nucleo è ancora eccitato perché i suoi protoni e neutroni non hanno ancora raggiunto la nuova situazione di equilibrio: di conseguenza, il nucleo si libera rapidamente del surplus di energia attraverso l'emissione di una radiazione gamma. Per esempio il cobalto-60 si trasforma

per disintegrazione beta in nichel-60, che raggiunge il suo stato di equilibrio emettendo una radiazione gamma Al contrario delle radiazioni alfa e beta, le radiazioni gamma sono molto penetranti e per bloccarle occorrono materiali ad elevata densità come il piombo. [2]

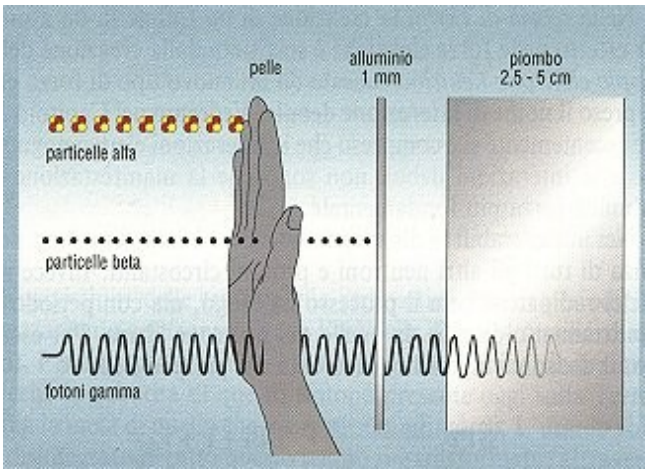


Fig. 2: Il potere penetrante delle diverse radiazioni

1.2. Monitoraggio

Lo smantellamento delle centrali nucleari e lo stoccaggio dei rifiuti radioattivi da esse prodotti, pone il problema del monitoraggio delle scorie a breve, medio e lungo termine. E' importante poter misurare il tasso di radioattività intorno a ciascun bidone di stoccaggio, indipendentemente dalla sua disposizione topologica e dalla sua struttura fisica ; in quanto ciò permetterebbe di verificare prontamente un danno alla struttura del bidone dovuto ad un eventuale cedimento strutturale.

L'ideale sarebbe prevedere una rilevazione in regime continuativo con sensori dedicati ad ogni fusto o gruppo di fusti, ciò può essere fatto utilizzando dei contatori Geiger-Muller[4]; questi dispositivi sono però molto costosi e bisogna quindi adottare altre soluzioni, come controlli periodici fatti da operatori oppure da robot appositi e sistemi di monitoraggio fissati all'interno dei depositi.

Le caratteristiche che deve avere un sensore per soddisfare le specifiche della problematica trattata sono: resistenza alle radiazioni, funzionamento da contatore,alta sensibilità, bassa efficienza, affidabilità, sensibilità alla posizione, robustezza meccanica, ridondanza, basso costo, facilità nell'installazione, manutenzione, rimozione e assemblaggio, sensibilità a specifiche radiazioni, capacità di misurare la dose totale, indipendentemente dalla registrazione storica dei conteggi riportati.

A questo scopo il progetto DMNR ha creato e testato un nuovo tipo di rivelatore di radiazioni ionizzanti, che risponde alle caratteristiche sopra citate e si comporta come un contatore Geiger-Muller (ma a differenza di questi risulta essere notevolmente più economico e versatile). Tale rivelatore può essere replicato a formare una griglia attorno ad ogni bidone con lo scopo di conteggiare la radiazione gamma.

Ciascun rivelatore consiste di una fibra ottica realizzata con scintillatore plastico, con accoppiati alle due estremità due fotosensori al silicio (SiPM) di ultima generazione, capaci di rivelare i deboli segnali luminosi prodotti dall'interazione con la radiazione incidente e convertirli in impulsi elettrici veloci.

[4] Un contatore Geiger-Muller è utilizzato per contare il numero di decadimenti che avvengono in un intervallo di tempo

In questo modo è possibile posizionare griglie di sensori intorno a ciascun bidone e registrare costantemente l'attività emessa, al fine di misurare in tempo reale il tasso di radiazione istantaneo. [3] [6]

Le fibre ottiche scintillanti, che rappresentano l'elemento sensibile di base, sono filamenti di materiale vetroso o polimerico che producono fotoni di luce visibile quando sono attraversati da radiazioni ionizzanti.

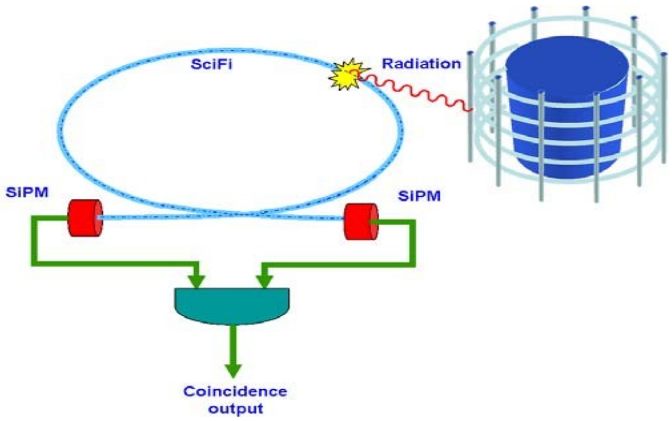


Fig. 3: Disposizione delle fibre attorno al fusto e coincidenza dei segnali elettrici ai capi di ogni fibra scintillante.

Per dimostrare la sensibilità della fibra scintillante sono stati condotti alcuni test tramite misure di precisione e simulazioni sottoponendo il sensore ad una sorgente con bassa attività come il ^{60}CO (cobalto-60)[5] o il ^{137}CS (cesio-137)[6].

Come si può osservare dai grafici che seguono, i risultati ottenuti mostrano come effettivamente la fibra sia sensibile alla radiazione gamma, infatti, in corrispondenza della sorgente è visibilmente maggiore il numero di gamma conteggiati[1].

[5] Il ^{60}CO è un isotopo radioattivo del metallo cobalto che decade per decadimento beta nell'isotopo stabile nichel-60.

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

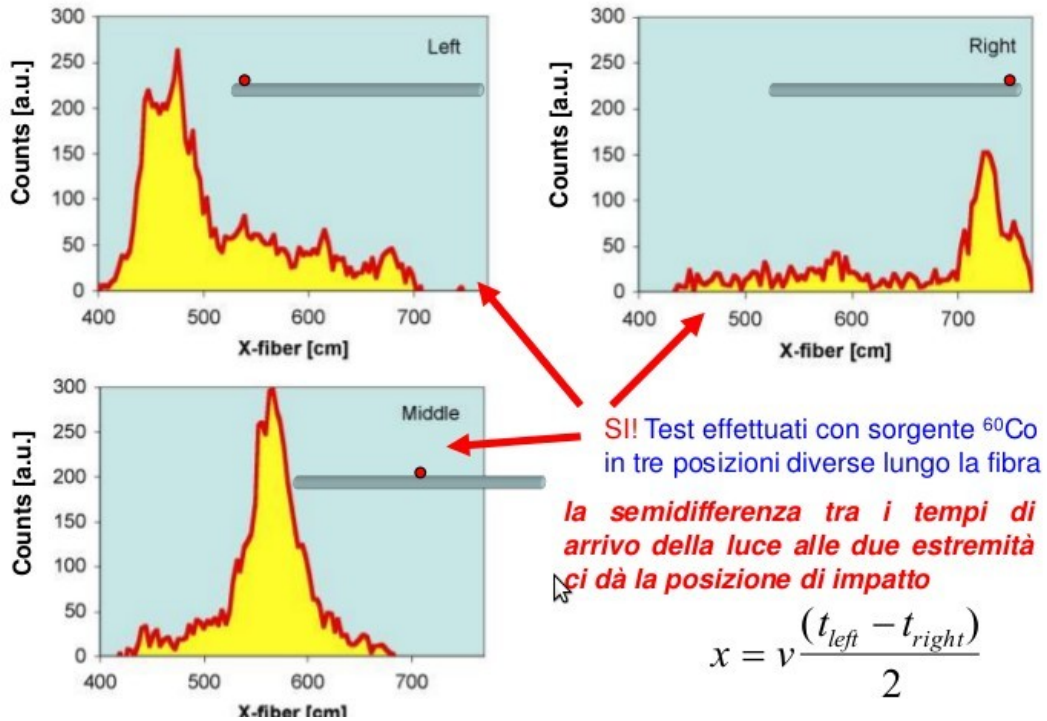


Fig. 4: Sensibilità della fibra e del SIMP

2. Metodo Monte Carlo

Il termine “Monte Carlo” si riferisce a tutti i metodi numerici che coinvolgono dei processi di campionamento statistico utilizzati al fine di approssimare le soluzioni di alcuni problemi, non solo di carattere probabilistico, ma anche deterministico, come, ad esempio, l’ottimizzazione e l’integrazione numerica.

Il capostipite di questo tipo di calcoli fu probabilmente il Conte di Buffon, che lanciando a caso un ago di lunghezza L sopra un tavolo su cui erano state disegnate due rette a distanza d (d > L) e calcolando la probabilità che l'ago intersecasse una delle due rette, riuscì a stimare il valore di π.

La storia moderna del Metodo Monte Carlo ebbe inizio negli anni ’40, 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, suscitò l’interesse di molti scienziati, che compresero subito che la potenza di calcolo poteva essere sfruttata per far eseguire alla macchina calcoli complessi come la risoluzione di equazioni differenziali.

Il primo a sfruttare le potenzialità del calcolatore ENIAC per lo studio delle reazioni termonucleari fu John Von Neumann, che a quel tempo lavorava al progetto Manhattan[4] per la costruzione della bomba atomica.

Agli sviluppi dell'ENIAC, che fu completato nella primavera del 1946, parteciparono diverse personalità scientifiche tra cui Enrico Fermi, Nicholas Metropolis, Edward Teller e in seguito Stanislaw Ulam. Quest'ultimo, grazie alle sue conoscenze matematiche, fu il primo che comprese le potenzialità dell’uso dei calcolatori per automatizzare il processo di campionamento statistico. Egli, con John Von Neumann e Nicolas Metropolis, sviluppò i primi algoritmi di campionamento statistico e scoprì come trasformare i processi descritti da equazioni differenziali deterministiche, in una forma equivalente interpretabile come una successione di operazioni stocastiche.

Essi contraddistinsero questi metodi col nome “Monte Carlo” data l’analogia tra la simulazione stocastica e il gioco d’azzardo praticato nel celebre casinò della città del Principato di Monaco.

Già, quindici anni, prima Enrico Fermi aveva applicato tecniche di campionamento statistico per studiare le proprietà del neutrone e fu di fatto il primo ad utilizzare questo metodo.

Oggi, con il termine metodo Monte Carlo, grazie al continuo sviluppo delle tecnologie informatiche, si fa riferimento ad classe di algoritmi computazionali usati per simulare il comportamento di sistemi fisici, matematici o di altri campi, che coinvolgano un gran numero di variabili.

L' esempio tipico della simulazione Monte Carlo è il metodo “hit or miss” , che è dovuto a Von Neumann e trova una delle sue più naturali applicazioni nella risoluzione degli integrali non risolvibili analiticamente.

Tale metodo può, quindi, 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. [5]

2.1. Simulazione Monte Carlo

La simulazione consiste nello studio del comportamento di un sistema mediante la sua riproduzione in un contesto controllabile.

Nella simulazione al calcolatore:

- si costruisce un modello matematico, costituito da equazioni che descrivono le relazioni tra le componenti del sistema ;
- si effettuano esperimenti "virtuali", assumendo che i risultati di tali esperimenti costituiscono una "riproduzione" sufficientemente accurata del comportamento che avrebbe il sistema;
- si verifica la validità d'ipotesi sul modello;
- si raccolgono informazioni per poter formulare possibili previsioni, per implementare meccanismi di controllo del sistema modellato.

Una particolare simulazione è la simulazione Monte Carlo, utilizzata per

- risolvere numericamente un problema in cui sono coinvolte anche variabili aleatorie, producendo un numero N sufficientemente elevato di possibili combinazioni dei valori che le variabili d'ingresso possono assumere (scenari) ,e calcolandone il relativo output sulla base delle equazioni del modello;
- testare più facilmente gli effetti di modificazione nelle variabili d'ingresso o nella funzione di output.

Tra i simulatori, che utilizzano il metodo Monte Carlo, vi sono:

- MCNP (Monte Carlo N-Particle Transport Code): è un package per la simulazione di processi nucleari; è sviluppato dal Los Alamos National Laboratory ed è distribuito negli Stati Uniti dal Radiation Safety Information Computational Center e in ambito internazionale dal Nuclear Energy Agency; è usato principalmente per la simulazione di processi nucleari, come la fissione, ma ha anche la capacità di simulare l'interazione di particelle che coinvolge neutroni, fotoni ed elettroni;

le aree di applicazione includono radiation protection e dosimetria, fisica medica, nuclear criticality safety, progetto di reattori di fusione, etc.

- EGS (Electron Gamma Shower): è un package general purpose per la simulazione Monte Carlo del trasporto di coppie di elettroni e neutroni; questo toolkit per la simulazione è scritto in linguaggio Mortran3; è stato sviluppato da A. James Cook e L. J. Shustek allo Stanford Linear Accelerator Center (SLAC) e tutte le sue caratteristiche sono state incluse in Geant3.
- 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 precompilati 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[7].
- GEANT (Geometry and Tracking) software di simulazione progettato per descrivere il passaggio di particelle elementari attraverso la materia utilizzando metodi Monte Carlo. 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 [19]. Superata una fase in cui non ha avuto una licenza software ben definita, a partire dalla versione 8.1 (rilasciata il 30 Giugno 2006) Geant4 è disponibile con la "Geant4 Software License". Attualmente nel sito web (http://geant4.cern.ch/) sono disponibili le versioni 4.9.4p04 rilasciata il 13 aprile 2012 e la 4.9.5 rilasciata il 23 maggio 2012.

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

3. GEANT

Geant è appunto un codice Monte Carlo creato nel 1974, che simula il passaggio di particelle attraverso la materia e fornisce un insieme completo di strumenti per tutte le aree di simulazione tipiche di queste applicazioni: geometria e risposta di rivelatori, tracciamento, gestione di run, evento e traccia, visualizzazione ed interfaccia utente. Inizialmente ideato per esperimenti alle alte energie, viene oggi applicato anche in ambito medico,biologico,spaziale e radioprotezionistico.

Le principali funzioni di Geant sono il trasporto di particelle attraverso l'apparato sperimentale, la simulazione della risposta dei rivelatori posti lungo la traiettoria e la rappresentazione grafica dell'apparato sperimentale e delle traiettorie delle particelle[22].

La prima versione di Geant è stata scritta nel 1974 e si trattava di un sistema di base che consentiva il trasporto solo di un piccolo numero i particelle attraverso dei rivelatori. Questa versione è stata sviluppata nel corso degli anni fino ad ottenere la versione Geant3, nata da un'idea di Renè Brun e Andy MaPherson nel 1982. Tale versione è in linguaggio Fortran.

Nel 1998 è uscita la versione Geant4, che è completamente nuova rispetto alle precedenti in quanto usa come linguaggio base di programmazione il C++, anziché il Fortran, utilizzando la tecnologia object-oriented.

3.1. Design e architettura Geant4

Geant4 [14] è l'acronimo di *GEometry ANd Tracking* (geometria e tracciamento) ed è stato proposto ed approvato dal DRDC (Detector Research and Development Committee) del Cern[7] alla fine del 1994. Il primo prototipo è nato alla fine del 1995 e la prima versione alpha risale alla primavera del 1997. Il software è stato sviluppato dalla collaborazione di un centinaio di scienziati provenienti da più di 40 istituti europei, americani, canadesi,russi e giapponesi. La fase di produzione di Geant4 è gestita da una collaborazione tra diversi gruppi sperimentali e laboratori[8], e durante le prove delle versioni alpha e beta del software, il codice è stato gestito e corretto dal progetto da RD44 [18].

La prima versione di produzione è stata pubblicata nel dicembre 1998; con questa RD44 ha concluso il suo mandato, ed è stata creata una nuova Collaborazione internazionale Geant4[19], regolata da un Memorandum of Understanding (MoU), per la gestione della fase di produzione. Il codice e la documentazione di Geant4 sono disponibili apertamente all'intera comunità scientifica sul Web [20].

Come accennato precedentemente Geant4 è interamente scritto in C++ ed utilizza la tecnologia Object-Oriented , che ha permesso di progettare le classi del toolkit in maniera estremamente riutilizzabile e di stabilire una corrispondenza chiara e personalizzata tra le particelle e i processi.

Geant4 fornisce un'ampia gamma di funzionalità necessarie per la simulazione dell'interazione delle particelle con la materia:

- permette di definire la geometria del sistema, i materiali coinvolti, le particelle e i loro processi fisici;
- permette di tracciare il moto delle particelle nella materia;
- consente di descrivere la risposta delle componenti sensibili di un rilevatore;
- fornisce strumenti di visualizzazione del rivelatore e delle traccie.

[7]Laboratorio Europeo per la Fisica delle Particelle Elementari
[8]Istituti che utilizzano Geant: SLAC, CERN, CEA, KEK, INFN, Manchester University STFC, etc...

Il seguente diagramma mostra le varie componenti di Geant4 e le dipendenze tra esse e come si evince da esso, il sistema è modulare, ciò permette all'utente di utilizzare solo le componenti di cui ha bisogno nella sua applicazione.

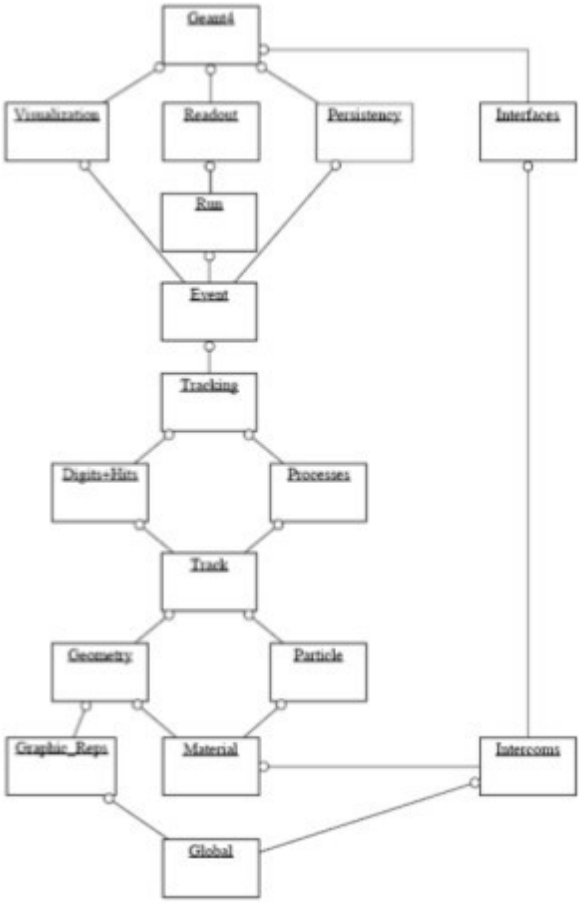


Fig. 5. Schema a blocchi di Geant4.

Le principali categorie usate per la simulazione del passaggio delle particelle nella materia sono:

- geometria e materiali (*Geometry, Materials*);
- gestione delle interazioni delle particelle con la materia (*Processes*);
- gestione della risposta del rivelatore (*Hit*);
- gestione delle traiettorie delle particelle (*Tracking*);
- gestione degli eventi (*Event, Run*);

- visualizzazione della geometria coinvolta e delle tracce delle particelle (*Visualization*);
- interfaccia utente (*User interfaces*).

Vediamo nel dettaglio tali categorie

Geometria

La categoria *Geometry* permette di descrivere l'apparato geometrico con accuratezza e di far navigare le particelle attraverso gli elementi geometrici in modo efficiente.

Un elemento geometrico è caratterizzato da tre diversi livelli:

- *solido* (G4Box, G4Tubs, etc.), ke contiene la definizione della forma e delle dimensioni;
- *volume logico* (G4LogicalVolume), dove vengono definiti gli attributi come il materiale di cui è composto, la presenza di eventuali campi elettromagnetici e la sensibilità del rilevatore;
- *volume fisico* (G4PVPlacement), che rappresenta il posizionamento del volume logico rispetto al volume madre che lo contiene. In questo modo si ha una struttura ad albero gerarchico, dove ogni volume contiene volumi più piccoli non sovrapponibili [15].

La geometria è definita attraverso un certo numero di volumi e quello più grande è detto *World Volume* (o volume madre). Esso non è contenuto all'interno di nessun altro volume, è posizionato nell'origine del sistema di coordinate globale e rappresenta il sistema di riferimento per tutti gli altri volumi creati in seguito al suo interno e che quindi saranno detti "volumi figli".

Materiali

Le due categorie *materials* e *particles* permettono di descrivere le proprietà fisiche dei materiali e delle particelle.

I materiali possono essere definiti attraverso le seguenti classi:

- *G4Element*, che descrive le proprietà degli atomi: numero atomico, numero di nucleoni, massa atomica, etc.;
- *G4Isotope*, che definisce i diversi isotopi in base al loro numero ed alla massa di una mole.
- *G4Material*, che descrive le proprietà macroscopiche della materia: densità, stato, temperatura, pressione etc.;

Le particelle, che possono essere definite in Geant4, sono classificate in:

- particelle elementari, cioè particelle che hanno tempo di volo finito e che possono essere distinte in: particelle stabili, ovvero che non decadono, come gamma, elettroni, protoni e neutroni; particelle a vita media-lunga, come muoni; particelle a vita media-breve; fotoni ottici; geantino[9].
- Ioni, che si dividono in ioni leggeri e pesanti
- quarks e Di-quarks.

Le particelle vengono definite usando le seguenti classi:

- *G4ParticleDefinition*, che ne descrive le proprietà (nome, massa, carica, etc.) e permette di associare a ciascuna particella la lista dei processi ai quali è sensibile;
- *G4Particle*, che permette di definire altre particelle ed ogni particella ha la propria classe che ne descrive le proprietà;
- *G4ParticleTable*, che praticamente è il dizionario delle particelle attualmente conosciute e permette di ricavare tutte le più importanti proprietà di esse.

Per configurare la proprietà di qualunque fascio di particelle si utilizza la classe *G4ParticleGun* [17], che permette di stabilire: il tipo di particella di cui è composto il fascio, l'energia iniziale della particella, la posizione di partenza del fascio e il momento di ogni particella.

[9] È una particella non reale, ideata dagli sviluppatori di Geant4 e priva di interazione, utile per testare il sistema.

Interazioni delle particelle con la materia

La categoria *processes* gestisce tutti i processi fisici che partecipano alle interazioni della particella con la materia.

Risposta del rilevatore

La categoria *Hit+Digitization* permette di definire quali e quante parti di un rilevatore siano sensibili e di gestire tali parti attraverso la generazione di *hit* e *digit* che costituiscono la risposta della parte sensibile del rivelatore (*sensitive detector*) alle particelle che lo attraversano.

Traiettoria delle particelle

La categoria *Tracking* si occupa della gestione delle traiettorie delle particelle attraverso gli step. Uno step è un oggetto che contiene sia le informazioni relative al passaggio di una particella da un punto (inizio step) ad un altro (fine step) sia le informazioni relative al transito della particella come, ad esempio, l'energia persa durante lo step e il tempo impiegato a percorrerlo. Quindi, una traiettoria è la registrazione della traccia di una particella nel suo complesso, ricavata dall'analisi degli step.

Gestione degli eventi

Le categorie *events* e *run* gestiscono rispettivamente gli eventi e una collezione di essi. Gli eventi vengono gestiti tramite la classe G4Event, tramite cui è possibile ottenere informazioni sulle traiettorie di tutte le particelle originate dall'evento.

Visualizzazione

La categoria *Visualization* permette di visualizzare la geometria del sistema, le tracce delle particelle e gli hits nei rivelatori. Essendo Geant4 compatibile con driver di differenti sistemi di grafica come, ad esempio, OPENGL, VRML,DAWN, etc.,

l'utente può controllare la corretta implementazione della geometria, potendo osservare il sistema da diversi punti di vista.

Interfaccia utente

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

L'utente può 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. Alcune di queste directory di comandi messe a disposizione dal toolkit sono:

- /run/: contiene i comandi legati allo svolgersi del run come *initialize*, per inizializzare il kernel di Geant4, *beamOn, per* far partire il run e *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* permette di memorizzare e visualizzare o no le traiettorie; *verbose* indica il livello di informazioni sulle tracce delle particelle che devono essere visualizzate.
- /particle/: contiene i comandi relativi alle particelle incidenti: *select* permette di selezionare una particella, *list* stampa la lista delle particelle disponibili in Geant, *find* permette di 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, etc.
- /gun/: gestisce i comandi del fascio incidente, il tipo di particella, la direzione, il punto di partenza, l'energia cinetica, ecc. Il comando *list* stampa la lista delle particelle disponibili; *number* definisce il numero di particelle da generare; *random* lancia in modo casuale la particella incidente.

- /gps/: gestisce i comandi per la generazione di sorgenti estese, il tipo di particella, la direzione, il punto di partenza, etc. Il comando *type* permette di definire il tipo di sorgente, mentre *shape* permette di definire la forma della sorgente.

3.2. Classi astratte e derivate

Partendo dalle classi astratte fornite dal kernel di Geant4, è possibile definire delle classi concrete. In Geant4 esistono due tipi di classi: quelle obbligatorie, che vengono utilizzate per inizializzare l'applicazione e l'esecuzione, e quelle opzionali, che permettono all'utente di modificare e personalizzare il comportamento di default di Geant4.

Le classi obbligatorie sono:

- *G4VUserDetectorConstruction*: nella classe derivata da questa classe astratta è necessario definire l'area di funzionamento della simulazione con la descrizione di tutte le geometrie, i materiali, le aree sensibili e le superfici.
- *G4VUserPhysicsList*: nella classe derivata da questa classe astratta è necessario definire tutte le particelle ed i processi fisici che interagiranno durante la simulazione.
- *G4VUserPrimaryGenerationAction*: nella classe derivata da questa è necessario specificare il modo in cui devono essere generate le particelle primarie.

L'esistenza di queste tre classi viene verificata all'inizio della simulazione dal *G4RunManager* al momento dell'invocazione dei metodi *initialize()* e *beamOn()*.

Le classi opzionali, invece, sono:

- *G4UserRunAction*: permette di impostare azioni da eseguire all'inizio ed alla fine di ogni run della simulazione.
- *G4UserStackingAction*: permette di personalizzare l'accesso allo stack in cui vengono memorizzate le informazioni di tracciamento delle particelle.

- *G4UserTrackingAction*: permette di impostare l'azione da eseguire all'inizio, alla creazione ed al completamento di ogni tracciato.
- *G4UserSteppingAction*: permette di personalizzare le azioni da eseguire ad ogni step del processo di simulazione.
- *G4UserEventAction*: per impostare azioni da eseguire all'inizio ed alla fine di ogni evento della simulazione.

4. Rilevatori a scintillazione

Un rivelatore di particelle è in generale uno strumento in grado di evidenziare il passaggio di una radiazione. Quando una particella (carica o neutra) attraversa la materia, interagendo con essa in modo elettromagnetico o nucleare, cede al mezzo tutta o parte della sua energia, determinando un eccitazione degli atomi (o molecole) con conseguente emissione di fotoni, o una ionizzazione: i fotoni e gli elettroni prodotti possono essere rivelati, consentendo di riconoscere il passaggio della particella.

Per una vasta categoria di rivelatori, possiamo affermare che il risultato finale dell'interazione è la presenza nel mezzo di un certo numero di cariche elettriche.

In realtà esistono vari tipi di rivelatori, che, oltre a segnalare il passaggio della particella, sono in grado di fornire ulteriori informazioni su: energia delle particelle che hanno attraversato il rivelatore, posizione, numero di particelle, quantità di moto, istante di attraversamento e identificazione delle particelle.

Una schematizzazione un po' semplicistica suddivide i diversi rivelatori in due categorie:

- i *contatori*, che sono in grado semplicemente di contare le particelle che attraversano il mezzo, senza fornire informazioni sulla loro traiettoria;
- i *rivelatori a visualizzazione*, che oltre a segnalare il passaggio delle particelle, permettono di osservarne la traccia e di misurarne le caratteristiche.

La prima considerazione da fare su un rivelatore è la sua capacità di produrre un segnale utile per un dato tipo di radiazione e di energia. Nessun rivelatore sarà sensibile a tutti i tipi di radiazione e a tutte le energie [9]. In realtà ogni rivelatore è progettato per essere sensibile ad un certo tipo di radiazione in un determinato intervallo di energie; al di fuori di tale regione, si ottiene generalmente un segnale inutilizzabile o una forte diminuzione dell'efficienza di rivelazione. Tale caratteristica dipende da diversi fattori: la sezione d'urto per le interazioni all'interno del rivelatore, la massa del rivelatore, il rumore prodotto, il materiale di protezione che circonda il volume sensibile del rivelatore.

La sezione d'urto e la massa del rivelatore determinano la probabilità che la radiazione incidente ceda tutta o parte della sua energia all'interno del rivelatore per ionizzazione.

Il minimo segnale rivelabile [9] dipende dal limite inferiore fissato dal rumore generato dal rivelatore stesso e dall'elettronica annessa. Il rumore si presenta come una fluttuazione nella tensione o nella corrente in uscita dal rivelatore ed è in ogni caso presente anche se non c'è nessuna radiazione incidente (fondo). Ovviamente il segnale prodotto dalla ionizzazione, per essere rivelato, dovrà essere maggiore del valore medio di rumore presente. Come convenzione, si definisce generalmente minimo rivelabile il valore del segnale pari al livello medio di rumore di fondo più tre volte la sua incertezza.

Un ulteriore limite è fornito inoltre dal materiale di protezione che ricopre la finestra di ingresso del rivelatore. A causa dell'assorbimento del mezzo di protezione, infatti, solo una radiazione con energia sufficiente per attraversare tale spessore di materiale può essere rivelata.

Le caratteristiche più significative di un rilevatore sono:

- *Efficienza di rivelazione*: la particella incidente può rilasciare tutta o parte della sua energia nel rivelatore oppure attraversarlo senza subire interazioni, con probabilità che variano a seconda del tipo di radiazione, della sua energia e delle dimensioni del rivelatore. [12]
- *Risoluzione spaziale*: per i rivelatori sensibili alla posizione, si definisce la risoluzione energetica come la minima distanza necessaria affinché lo strumento sia in grado di distinguere due particelle che lo attraversano simultaneamente. [13]
- *Risoluzione energetica*: viene definita come la minima differenza di energia necessaria affinché il rivelatore riesca a discriminare due particelle di energia diversa. [12]
- *Risoluzione temporale*: è il minimo intervallo di tempo necessario al rivelatore per distinguere due particelle che lo attraversano in tempi diversi.

Se due particelle giungono ad intervalli di tempo minori, i rispettivi segnali si sommano generando il problema noto come “pile up”. [9]

- *Tempo morto*: è definito come l'intervallo di tempo che deve trascorrere dopo il passaggio di una particella, affinché lo strumento sia in grado di rivelarne una successiva. [12]

4.1. Scintillatori

Il rivelatore a scintillazione è un materiale che emette impulsi di luce quando viene attraversato da una radiazione ionizzante (come fotoni di alta energia o particelle cariche). Esso presenta molti vantaggi rispetto ad altri tipi di rivelatori, quali il veloce tempo di risposta e un'alta efficienza di rivelazione.

Uno scintillatore consiste generalmente in un materiale scintillante accoppiato otticamente ad un fotomoltiplicatore; quando la particella passa attraverso lo scintillatore eccita gli atomi e le molecole dello scintillatore, emettendo la luce che viene trasmessa al fotomoltiplicatore e viene convertita in una debole corrente di fotoelettroni. Viene così generato un impulso elettrico che potrà essere elaborato per condurre analisi e ottenere informazioni sulla particella che ha colpito lo scintillatore stesso[11].

Gli scintillatori utilizzati per la rivelazione vengono generalmente suddivisi in due categorie a seconda del materiale di cui sono composti: *scintillatori organici* ed *inorganici*. Gli organici danno generalmente una risposta veloce entro i 10ns (2ns per i plastici) ma producono poca luce (~ 4000 eV) e hanno una bassa efficienza di rivelazione; sono quindi più indicati per misure di tempo. Gli inorganici, tra cui ricordiamo ad esempio i cristalli, sono invece più lenti con tempi tipici di risposta di 200 500ns, ma hanno una maggiore luminosità (38000 eV nel caso dello ioduro di sodio) risultando quindi più indicati per misure di energie.

4.2. Fibra scintillante

La classe degli scintillatori a fibre ha il pregio notevole di permettere di ottenere una buona risoluzione spaziale. Esse sono sottili tubicini di polimero plastico, solitamente polistirene, drogato con materiale scintillante.
Funzionano come guide di luce: la radiazione prodotta dall'eccitazione conseguente al passaggio di una particella è incanalata all'interno della fibra per mezzo di riflessioni contro un rivestimento (core cladding) con indice di rifrazione minore di quello del polimero plastico. Il core cladding spesso viene rivestito con un ulteriore strato, si ha in questo caso una fibra di tipo double-clad. Il diametro totale della fibra è compreso fra 0.5 mm e 1 mm. A differenza degli altri scintillatori spesso si ha un accoppiamento diretto al fotomoltiplicatore, senza guida di luce. È possibile avere perdite di segnale luminoso nel tragitto compiuto a partire dal suo punto di generazione: qualora la riflessione interna non sia totale, parte dell'energia è persa prima di giungere a destinazione.
Si possono utilizzare fibre ottiche con caratteristiche diverse a seconda della necessità, in base all'efficienza, distribuzione spettrale e temporale in quanto queste dipendono dal materiale utilizzato e dalla sua composizione.

Le fibre sono caratterizzate da flessibilità, immunità ai disturbi elettrici e alle condizioni atmosferiche più estreme (es. variazione di temperatura) ed è per queste loro caratteristiche che si è deciso di utilizzarle nel progetto DMNR. Infatti, ogni fusto del sito sarà circondato da una griglia di questi sensori, che costantemente misureranno la radiazione proveniente dai fusti stessi (si tratta principalmente di radiazioni gamma).

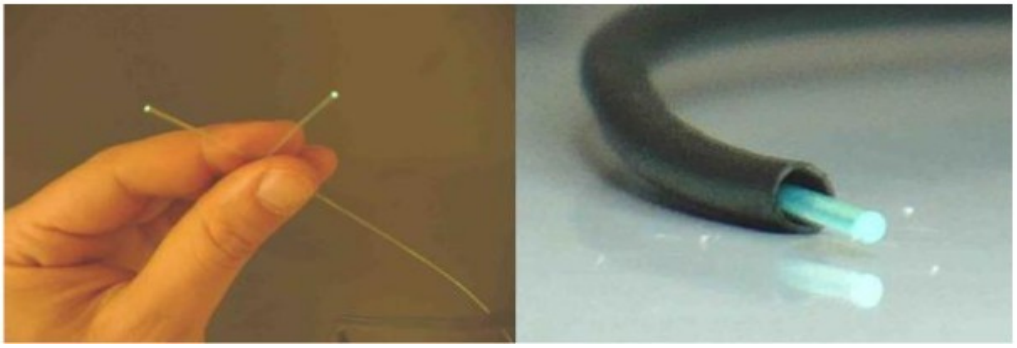


Fig. 6 : Una fibra scintillante. A destra è mostrata all'interno di una guaina nera che serve come protezione meccanica.

5.Sviluppo dell'applicazione

Il prototipo che è stato sviluppato riguarda la simulazione della rivelazione della radiazione gamma prodotta da sorgenti radioattive estese attraverso l'uso di geometrie di fibre scintillanti, in particolare sono state implementate le seguenti geometrie:

- sorgente estesa con tre fusti;
- sorgente estesa con sei fusti;
- sorgente estesa con nove fusti.

Si è deciso di implementare queste geometrie in quanto si è supposto che nel sito di stoccaggio i fusti siano sovrapposti fino a un massimo di tre e quindi le simulazioni sono state effettuate considerando inizialmente una sola pila (tre fusti), poi due pile (sei fusti) e in fine tre pile (nove fusti).

In seguito, sarà posta l'attenzione sull'architettura generale dell'applicazione e sulle classi di maggiore interesse definite per la costruzione delle diverse geometrie.

Durante la creazione del software di simulazione è stata utilizzata la visualizzazione grafica della geometria, in modo da verificare la corrispondenza tra la geometria progettata e quella reale. Tale modalità grafica è stata disattivata, durante le simulazioni vere e proprie, in modo da aggravare il meno possibile sul carico di lavoro della CPU, con il conseguente abbreviamento dei tempi di simulazione.

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

5.1. Main

Geant4 non fornisce un metodo main() ma occorre implementarlo sulla base delle proprie esigenze. Innanzitutto occorre creare le istanze delle due classi *manager* che gestiscono l'inizializzazione dell'applicazione:
G4RunManager è la classe manager preposta al controllo del flusso della simulazione attraverso la gestione del susseguirsi degli eventi all'interno di ciascun run ed è inoltre responsabile dell'inizializzazione dei parametri della simulazione. A questo scopo occorre fornire al *runManager* le informazioni necessarie per configurare i run, cioè: la struttura del detector (in termini di geometria, materiali e sensibilità di rivelazione), le particelle e i processi fisici da simulare, il setup dell'evento primario. Quanto appena detto, è definito nel codice sottostante mediante l'inizializzazione *runManager* e delle mandatory user class *DetectorConstruction*, *PhysicsList* e *PrimaryGeneratorAction*.

```
G4RunManager * runManager = new G4RunManager();
G4VUserDetectorConstruction* detector = new DetectorConstruction();
runManager->SetUserInitialization(detector);
G4VUserPhysicsList* physics = new PhysicsList();
runManager->SetUserInitialization(physics);
G4VUserPrimaryGeneratorAction*gen_action=new PrimaryGeneratorAction();
 runManager->SetUserAction(gen_action);
```

Fig. 7: Creazione dell'istanza G4RunManager e inizializzazione delle mandatory user classes

Le informazioni relative al generatore di particelle, racchiuse nella user action class *PrimaryGeneratorAction* vengono passate al *runManager* attraverso il metodo *SetUserAction()* come nella Fig. precedente. 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.

5.2. Detector Constuction

La classe *DetectorConstruction* contiene la definizione della geometria del detector e dei fusti, partendo dal volume madre che è il contenitore principale all'interno del quale verranno posizionati i volumi del dispositivo da realizzare.

Essa deriva la mandatory user class *G4VUserDetectorConstruction* del codice Geant4 che contiene il metodo virtuale *Construct()* che viene implementato nella *DetectorConstruction.cc* e che sarà invocato dal *runManager* durante l'inizializzazione dell'applicazione. All'interno di questo metodo deve essere inserito il codice per la definizione della geometria nonché del volume "world". Il metodo dovrà restituire il puntatore all'istanza del volume fisico che rappresenta l'intero rivelatore. Di seguito viene riportato il contenuto del file *DetectorConstruction.hh* che contiene la dichiarazione dei metodi e degli attributi della classe.

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

```
 ….....................
 // Parameters for detector
 G4double fibraLen;
 G4double fibraRadius;
 G4ThreeVector posFibra;
  G4double TubsLen;
};
```

Fig. 8: classe DetectorConstruction.hh

Nella classe *DetectorConstruction.cc* vengono definiti le caratteristiche del volume *world*, del fusto e del rilevatore. Vediamo ciò nel particolare.

Per definire le dimensioni del volume world, del fusto e del rilevatore viene usto il metodo *ComputeParameters()*.

```
void DetectorConstruction::ComputeParameters()
{
 //This function defines the defaults
 //of the geometry construction
 // ** world **
 halfWorldLength = 10.0*m;
 //** SOURCE **
 TubsLen = 0.5*m;
 // ** Fibra **
 fibraLen = 1.2*m;
 fibraRadius = 0.5*mm;
 posFibra = G4ThreeVector(0,0,0);
}
```

Fig. 9: DetectorConstruction.cc

Per definire il fusto viene usato il metodo *ConstructTubs()*, che utilizza la classe *G4Tubs* per costruire il fusto e riempirlo con il materiale *concrete* (cemento) definito nel metodo *DefineMaterials()*.

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

```
                                        G4ThreeVector(0,0,0),  //translation
                                        TubsLogic, //its logical volume
                                        "TubsCore", //its name
                                        logicWorld, //its mother volume
                                        false,
                                        1);
…....
}
```
Fig. 10: metodo ConstructorTubs() per la costruzione del fusto.

Per definire il rivelatore, che è costituito da quattro fibre, composte da un cladding di plexiglass e da un core di polistirene, poste attorno a ciascun fusto viene usato il metodo *ConstructFibra()*. Tale metodo che permette di definire il core della fibra, utilizzando la classe *G4Tubs* e di riempierlo con il materiale polystyrene definito nel metodo *DefineMaterials()*, di istanziare gli oggetti necessari per costruire il cladding costituito da plexiglass, di definire il volume sensibile, ovvero l'oggetto *sensitive detector* sensibile al passaggio delle particelle. Definiti i volumi, si crea l'oggetto *sensitive* della classe *FibraSensitiveDetector* e si registra tramite l'oggetto gestore *SDMan* (*G4SDManager*) attraverso il metodo *AddNewDetector()*. Si invoca il metodo *SetSensitiveDetector()* del corrispondente volume logico che si vuole rendere sensibile (il core), passandogli come parametro l'oggetto *sensitive* precedentemente istanziato e registrato tramite il *Sensitive Detector Manager*. Inoltre, viene utilizzata una matrice di rotazione per ruotare in maniera appropriata le fibre.

```
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);
// 2nd: definire l'oggetto logico
G4LogicalVolume* fibraCladdingLogic = new G4LogicalVolume( fibraCladdingSolid,//its solid
                                                           plexiglass,//its material
                                                           "fibraCladdingLogic"); //its name
// 3rd: definire l'oggetto
physiFibraCladding = new G4PVPlacement( pRot,// rotazione
```

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

Fig. 11: Metodo ConstructorFibra()per la costruzione e la sensibilizzazione della fibra.

5.3. Fibra Sensitive Detector

La classe *FibraSensitiveDetector c*ontiene i metodi per definire la risposta del rivelatore, infatti essa gestisce una collezione di oggetti, detti *hit*, che rappresentano un'istantanea dell'interazione fisica di una traccia nella regione sensibile del detector.

Essa deriva la classe astratta *G4VSensitiveDetector* del codice Geant4 e ne ridefinisce i metodi *Initialize()*, *Process Hits()*, *EndOfEvent()*, che gestiscono gli *hit* e la *hit collection*. Quest'ultima rappresenta una collezione di oggetti di tipo *FibraHit,* che è istanziata sulla base della template class *G4THitsCollection* del codice Geant4 attraverso una procedura standard riportata di seguito.

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

Fig.12: Classe FibraHit: assegnazione del nome FibraHitCollection al tipo di dato G4THitsCollection.

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

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

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

Fig. 13: Metodo costruttore della classe FibraSensitiveDetector .

Successivamente nel metodo *Initialize()* viene istanziato l'oggetto vero e proprio.

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

Fig. 14: Metodo Initialize() di FibraSensitiveDetector.

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*. La gestione della collezione di hit avviene attraverso la classe *FibraSensitiveDetector*, che seleziona gli oggetti *hit* da collezionare attraverso il metodo *ProcessHits()*.

5.4. Primary Generator Action

La classe *PrimaryGeneratorAction* è implementata derivando la *mandatory* user class *G4VUserPrimaryGeneratorAction* del codice Geant4 e contiene tutte le caratteristiche che riguardano il setup dell'evento primario (il tipo di particella che deve essere generata, la posizione, il momento, l'energia, etc) e che sono racchiuse nell'oggetto *G4ParticleGun* [14].

La *G4VUserPrimaryGeneratorAction* contiene il metodo virtuale *GeneratePrimaries()*, implementato per la generazione dell'evento primario ed avviato attraverso il metodo *generatePrimaryVertex()* del *G4ParticleGun*, che permette di generare una particella primaria di una certa energia da un certo punto a certo tempo ad una certa direzione.

Il metodo *G4GeneralParticleSource* [16] permette di generare le primary vertex a caso sulla superficie di un qualunque volume, la direzione di moto e l'energia cinetica può anche essere random. La distribuzione(isotropica, SolidAngle...) può essere impostata dai comandi dell'interfaccia utente.

```
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);
     G4String getSourceType();
     G4String getDistributionType();
     G4double getDistanceY();
private:
      G4ParticleGun* gun;
      PrimaryGeneratorMessenger* gunMessenger;
      G4double distanceY;
      G4String distribType;
      G4ThreeVector direct;
      G4String source;
};
```

Fig. 15: Metodi e attributi della classe PrimaryGeneratorAction.

Le simulazioni da noi effettuate sono state realizzate utilizzando una sorgente estesa isotropica realizzata implementando il *G4GeneralParticleSource,* che è parte del toolkit Geant4 per le simulazioni ad alta energia di trasporto delle particelle. In particolare, consente le specifiche spettrali, la distribuzione spaziale e angolare delle particelle primarie.

```
gun = new  G4GeneralParticleSource();
particle = G4ParticleTable:: getParticleTable()->findParticle("gamma");
```

Fig. 16: codice sorgente estesa

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

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

```
/gps/pos/halfy 0.4 m
/gps/pos/halfz 0.4 m
/gps/ang/type iso
/run/beamOn 100000
.....
```

Fig. 17: Macro file utilizzato per l'implementazione di una sorgente estesa di forma cilindrica.

5.5. Primary Generator Messenger

La classe *PrimaryGeneratorMessenge*r é implementata derivando la *G4UImessenger* del codice Geant4 e fa da tramite tra l'interfaccia utente e le classi contenenti il codice per l'esecuzione dei comandi [15]. Essa, infatti, contiene la definizione degli oggetti per i comandi e implementa il metodo *SetNewValue() per* definire il comportamento da associare ai comandi aggiunti.

Per raggruppare i comandi nella directory "*Fibra*", all'interno della quale sono stati posti i comandi aggiuntivi implementati, è stato istanziato un oggetto della classe *G4UIdirectory* del toolkit.

Gli oggetti istanziati per la definizione dei comandi sono *distrCmd*, *sourceCmd, sourceType*, a cui corrispondono i comandi *setDistrib*, *setDistance* e *setSourceType*, che permettono di settare rispettivamente tipo di distribuzione delle particelle irradiate, la distanza della sorgente dal rivelatore e il tipo di sorgente.

```
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);
 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);
```

```
}
PrimaryGeneratorMessenger::~PrimaryGeneratorMessenger(){
delete distrCmd;
delete sourceCmd;
delete gunDir;
}
```

Fig. 18: classe PrimaryGeneratorMessenger.

5.6. Run Action

La classe *RunAction* è implementata derivando la classe virtuale *G4UserRunAction*, che contiene alcuni metodi virtuali che saranno invocati dal *G4RunManager* durante il corso di un run:

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

Nel metodo costruttore avviene l'assegnazione del puntatore all'oggetto *action* della classe *PrimaryGeneratorAction* ricevuto come parametro, che servirà per accedere ai metodi get della classe. Ciò consentirà di ottenere i valori utili al resoconto della simulazione che verrà stampato a video e su file.
Nella classe *RunAction* sviluppata per l'applicazione sono stati ridefiniti i metodi *BeginOfRunAction()*, in cui avviene l'inizializzazione delle variabili relative al conteggio degli *hit,* e *EndOfRunAction()*, che è eseguito alla fine di ogni run e contiene il codice per salvare le informazioni relative al run appena eseguito nel file di testo "*output.txt*". Questo file contiene un resoconto sull'esito dei run eseguiti.

```
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;
}
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> Source Type : " << action->getSourceType() <<flush;
outFile << "\n> Hits count fibra1: " << sd->hitCount << flush;
outFile << "\n> Hits count fibra2: " << sd2->hitCount2 << flush;
outFile << "\n> Hits count fibra3: " << sd3->hitCount3 << flush;
outFile << "\n> Hits count fibra4: " << sd4->hitCount4 << flush;
outFile << "\n#####################################\n" << flush;
outFile.close();
G4cout << "\n> Source Type : " << action->getSourceType() << G4endl;
G4cout << "> Hits count fibra1: " << sd->hitCount << G4endl;
G4cout << "> Hits count fibra2: " << sd2->hitCount2 << G4endl;
G4cout << "> Hits count fibra3: " << sd3->hitCount3 << G4endl;
G4cout << "> Hits count fibra4: " << sd4->hitCount4 << G4endl;
G4cout << "\n#####################################\n" << G4endl;
G4String nomeFile = "spettro";
stringstream ss;
ss << aRun->GetRunID();
nomeFile.append(ss.str());
nomeFile.append(".txt");
d->scrivi_su_file(nomeFile);
}
```

Fig. 19: Metodi BeginOfRunAction() e EndOfRunAction() della classe RunAction.

5.7. Physics List

La classe PhysicsList, che deriva la mandatory user class *G4VUserPhysicsList,*definisce al suo interno i processi fisici e le particelle che intervengono nelle simulazioni.

Essa contiene tre metodi virtuali da implementare: *ConstructProcess()* e *ConstructParticle()*, che contengono la definizione dei processi fisici e delle particelle, e *SetCuts()*,che serve per impostare i valori di cut-off per le particelle definite.

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

Fig. 20: PhisycList.hh attributi e metodi della classe PhisycList

Geant4 fornisce vari tipi di particelle, ciascuna rappresentata da una classe derivata dall'interfaccia *G4ParticleDefinition*.

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

5.8. Compilazione e variabili d'ambiente

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

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

Fig. 21: Comandi per impostare le variabili d'ambiente prima di compilare o eseguire codice Geant4.

La variabile d'ambiente G4WORKDIR deve contenere il percorso alla directory di lavoro dove sono contenuti i file sorgenti delle classi, per comodità si lancia un file setupgeant.sh che permette l'esportazione di tutte le variabili. Come vediamo si esegue l'exporting delle librerie CLHEP che viene stabilito durante la fase di installazione del toolkit, si esegue lo script env.sh fornito dal toolkit, che imposta tutte le altre variabili d'ambiente necessarie e l'exoprting delle Qt che favoriscono lo sviluppo di un'esecuzione interattiva.
Dopo aver impostato le variabili d'ambiente si può compilare il programma invocando il comando make che genera il file eseguibile nella directory $G4WORKDIR/bin/$G4SYSTEM.

6. Simulazioni

Le simulazioni sono uno strumento sperimentale di analisi molto potente, utilizzato in molti ambiti scientifici e tecnologici dettato dalla difficoltà o impossibilità di riprodurre fisicamente in laboratorio reale le effettive condizioni da studiare e che si avvale delle grandi possibilità di calcolo offerte dall'informatica e dai sistemi di elaborazione. La simulazione, infatti, altro non è che la trasposizione di un "modello concettuale" della realtà.

Come già detto, si è deciso di effettuare le simulazioni considerando tre diverse geometrie:

- geometria con tre fusti
- geometria con sei fusti
- geometria con nove fusti

in quanto si è supposto che nel sito di stoccaggio i fusti siano sovrapposti fino a un massimo di tre e quindi le simulazioni sono state effettuate considerando inizialmente una sola pila (tre fusti), poi due pile (sei fusti) e in fine tre pile (nove fusti). A tali geometrie è stata applicata una sorgente estesa, che genera una distribuzione random di eventi in tutto lo spazio.

In fase di simulazione si è deciso di utilizzare una sorgente estesa di forma cilindrica

```
/gps/particle gamma
/gps/pos/type Volume
/gps/pos/shape Cylinder
/gps/pos/centre 0. 0. 0. cm
/gps/pos/radius 0.5 m
/gps/pos/halfx 0.4 m
/gps/pos/halfy 0.4 m
/gps/pos/halfz 0.4 m
/gps/ang/type iso
/run/beamOn 1000000
```

Fig. 22: file macro per la generazione della sorgente estesa

In tutte e tre le geometrie verranno lanciati un totale di 1.000.000 di eventi, in quanto, essendo il diametro della fibra 1mm, si è visto che, utilizzando un numero inferiore di eventi, la probabilità che la fibra venga colpita risulta essere molto bassa. Inoltre, è stata utilizzata una sorgente estesa isotropica ed essendo essa una distribuzione random di eventi, ad ogni run varierà il punto di emissione dell'evento stesso.

Nelle diverse figure che vedremo di seguito, sarà possibile distinguere i fusti di contenimento (di colore grigio), all'interno dei quali vi sarà la sorgente estesa (di colore verde), e i dispositivi di conteggio, cioè le fibre (di colore rosso tratteggiato), disposte attorno al fusto. Si è scelto di circondare ciascun fusto con quattro fibre, disposte orizzontalmente rispetto ad esso.
Vediamo nel dettaglio tali geometrie.

6.1. Geometria con tre fusti

In tale geometria sono stati presi in considerazione tre fusti sovrapposti l'uno sull'altro e ciascun fusto è circondato da quattro fibre. Le figure che seguono mostrano tale geometria e in particolare la Fig.24 mostra tale geometria nel caso in cui si ha la sorgente estesa che emette raggi gamma dal fusto centrale.

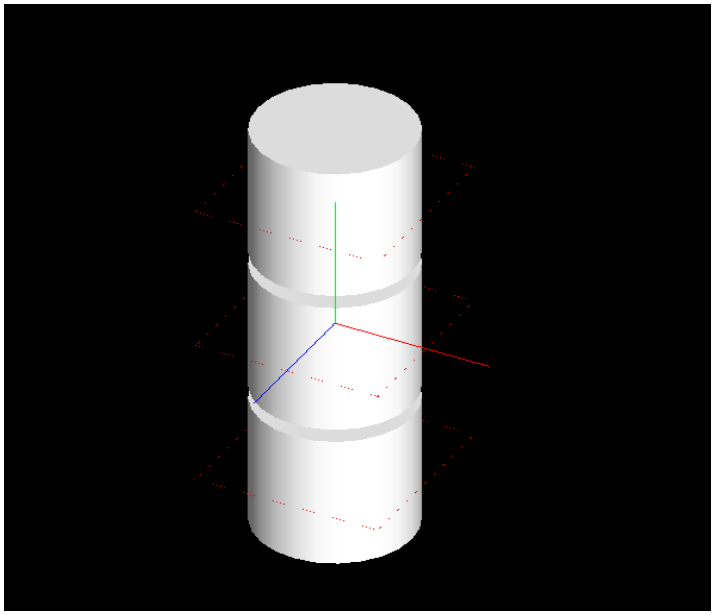


Fig. 23: Geometria con tre fusti

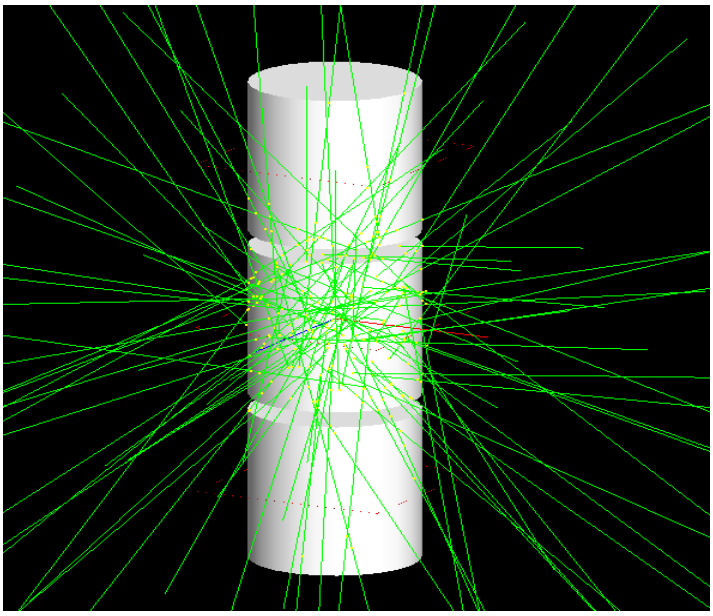


Fig. 24: Geometria con tre fusti con sorgente emessa dal fusto centrale

6.2. Geometria con sei fusti

In tale geometria sono stati presi in considerazione due pile di fusti con tre bidoni e ciascun fusto è circondato da quattro fibre. Le figure che seguono mostrano tale geometria e in particolare la Fig.26 mostra tale geometria nel caso in cui si ha la sorgente estesa che emette raggi gamma dal fusto in alto della pila di sinistra.

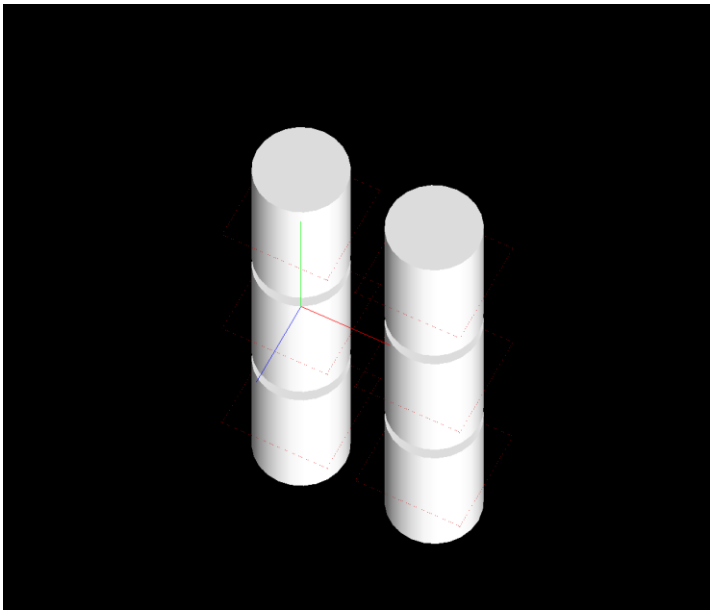


Fig. 25: Geometria con sei fusti.

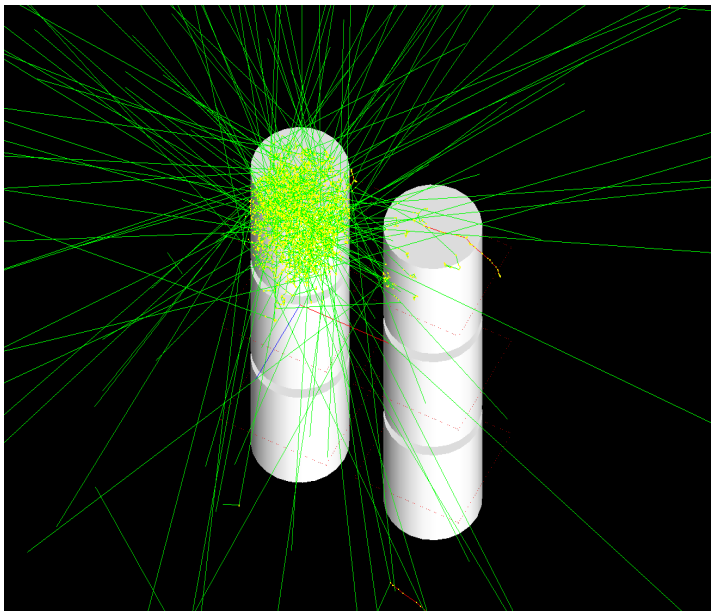


Fig. 26: Geometria con sei fusti con sorgente emessa dal fusto in alto della pila di sinistra

6.3. Geometria con nove fusti

In tale geometria sono stati presi in considerazione tre pile di fusti con tre fusti ciascuna e ciascun fusto è circondato da quattro fibre. Le figure che seguono mostrano tale geometria e in particolare la Fig.28 mostra tale geometria nel caso in cui si ha la sorgente estesa che emette raggi gamma dal fusto in basso della pila di sinistra.

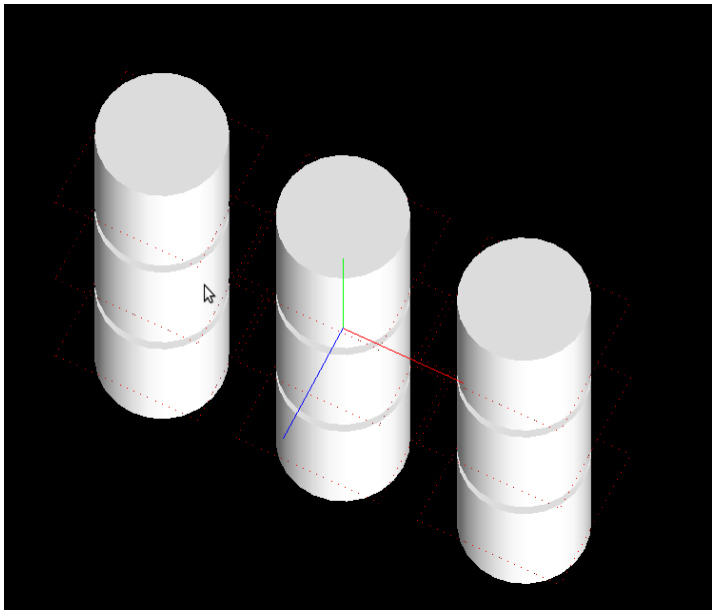


Fig. 27: Geometria con nove fusti

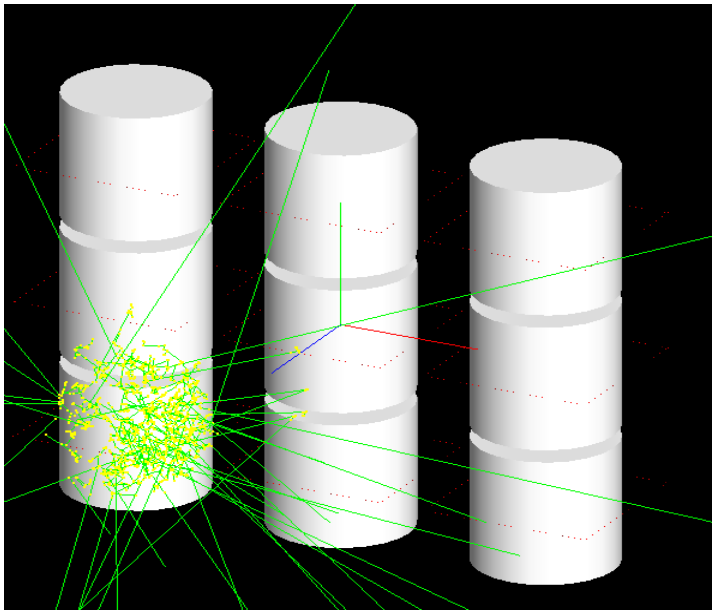


Fig. 28: Geometria con nove fusti con sorgente emessa dal fusto in basso della pila di sinistra

7. Risultati ottenuti

In tale capitolo ci occuperemo di verificare l'attendibilità delle simulazione effettuate analizzando i risultati ottenuti, che saranno dovuti alle particelle rilevate dal detector a meno di un 6% dovuto alle particelle che colpiscono il cladding.

L'obiettivo è quello di capire qual è il fusto danneggiato e ciò è possibile andando a vedere quali fibre hanno rilevato il maggior numero di eventi.

7.1 Verifica analitica

Prima di iniziare con le simulazioni si è attuata una verifica analitica della geometria del sistema, in modo da poter confrontare in seguito i valori ottenuti tramite tale verifica con quelli ottenuti dalle simulazioni.

Per verificare l'efficienza geometrica è stato calcolato il valore dell'angolo solido del rivelatore visto dalla sorgente. Per definizione l'angolo solido[21] 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, un la sua normale, dS0 la sua funzione ortogonale e u r il versore del raggio r uscente da un punto di osservazione O. (Fig.)

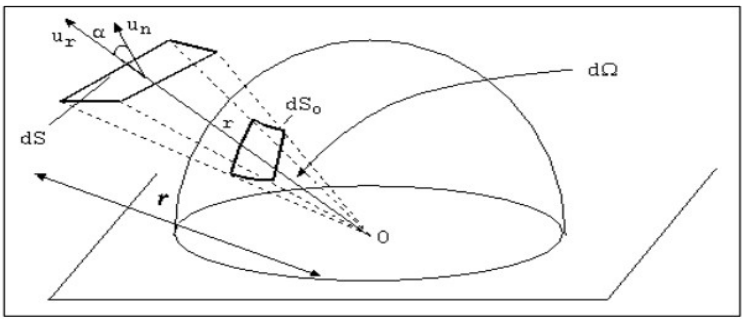


Fig. 29: Definizione di angolo solido.

Si definisce angolo solido infinitesimo la quantità:

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

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

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

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

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

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

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

Ponendo $\Theta_0=0$ 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:

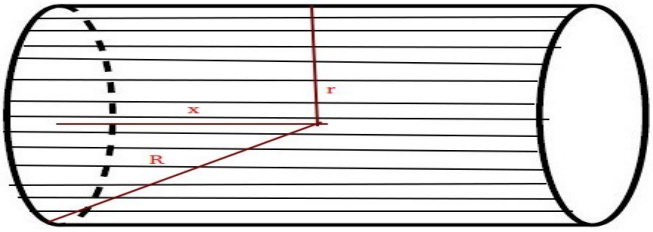
$$\varepsilon_g = \frac{area\ rivelatore}{area\ della\ sfera\ di\ raggio\ d} = \frac{A_{riv}}{4\pi d^2} = \frac{\Omega_{riv}}{\Omega_{tot}}$$

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

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

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 prove, senza visualizzazioni grafiche, allo scopo di sfruttare al massimo le prestazioni per consentire di generare un gran numero di eventi impiegando breve tempo. Le prove sono state eseguite generando le pseudo-particelle (geantini) in maniera isotropica in tutto lo spazio. Il rapporto tra il numero di particelle incidenti nel primo caso e il numero totale di particelle emesse è risultato coerente con il valore atteso per una fibra di lunghezza infinita ottenuto analiticamente.

In realtà il calcolo analitico della porzione di angolo solido è stato semplificato nel seguente modo. Consideriamo un cilindro costituito da un certo numero n di fibre (n_{fibra}) costruito a partire dalla stessa fibra, gli eventi contati dalla fibra, cioè $\varepsilon_{geometrico}$, saranno dati dall'angolo solido della fibra stessa Ω_{fibra} meno l'angolo solido delle calotte sferiche, $\Omega_{calotte}$, costruite sulle estremità cilindrica.



Dove x è la lunghezza della fibra ed r è il raggio, rispettivamente pari a 0.60 m e 0.05 cm. Pertanto i valori analitici sono stati calcolati con i seguenti passi:

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

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

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

determiniamo il numero delle fibre contenute nel cilindro

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

calcoliamo adesso l'angolo solido della fibra:

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

il valore dell'efficienza geometrica è determinato da:

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

quindi il risultato atteso per la nostra implementazione con un valore di 1.000.000 di eventi generati dalla sorgente è:

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

a bontà del risultato analitico è determinato da:

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

7.2 Risultati ottenuti con la geometria a tre fusti

Di seguito viene riportato il file di output nel caso si simuli una geometria con tre fusti e la sorgente estesa venga emessa dal fusto centrale.

Run
> Source Type : Cs137
> Hits count fibra1: 169
> Hits count fibra2: 210
> Hits count fibra3: 210
> Hits count fibra4: 182
> Hits count fibra5: 54
> Hits count fibra6: 43
…....
####################################

Fig. 30: Porzione dei risultati ottenuti con la geometria a tre fusti

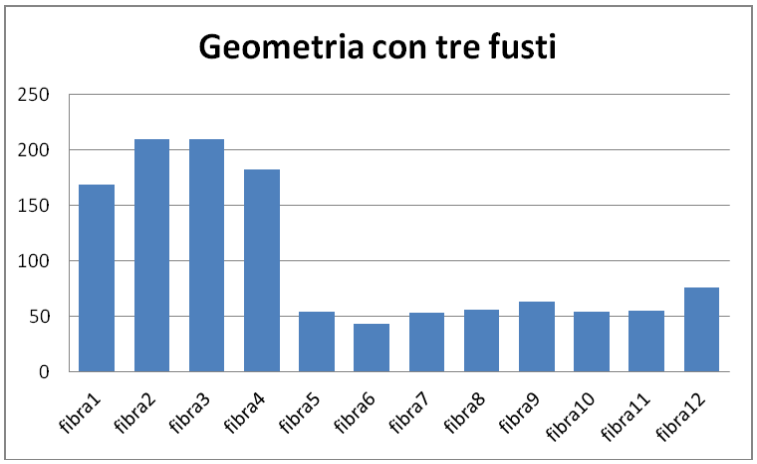


Fig. 31: Risultati ottenuti quando la sorgente estesa emette dal fusto centrale.

Come si osserva dal grafico, le fibre del fusto danneggiato contano un numero di eventi maggiore.

7.3. Risultati ottenuti con la geometria a sei fusti

Di seguito viene riportato il file di output nel caso si simuli una geometria con sei fusti e la sorgente estesa venga emessa dal fusto in alto della pila di destra.

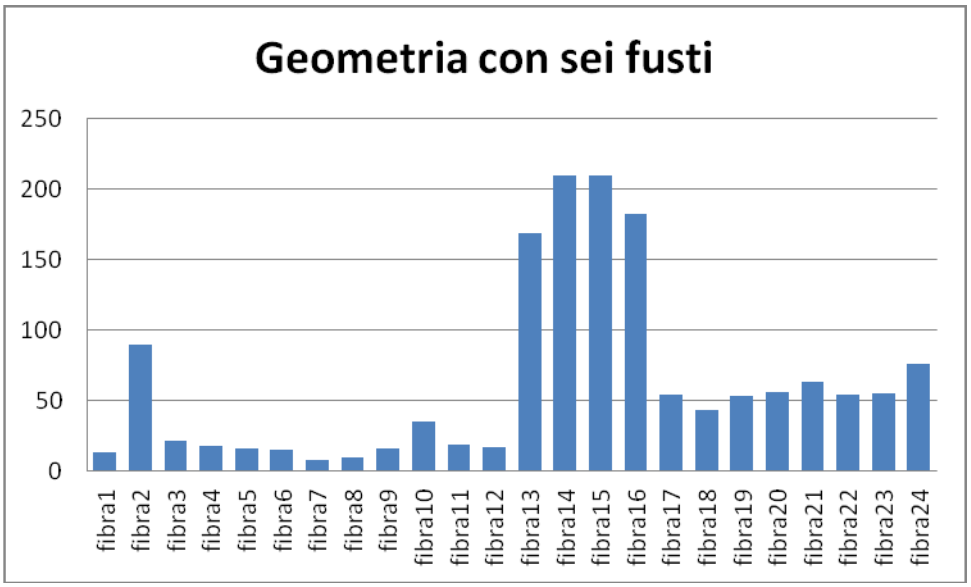


Fig. 32: Risultati ottenuti quando la sorgente estesa emette dal fusto in alto della pila di destra.

Come si osserva dal grafico, le fibre del fusto danneggiato contano un numero di eventi maggiore.

7.4. Risultati ottenuti con la geometria a nove fusti

Di seguito viene riportato il grafico che si ottiene nel caso si simuli una geometria con nove fusti e la sorgente estesa venga emessa dal fusto centrale della pila di destra.

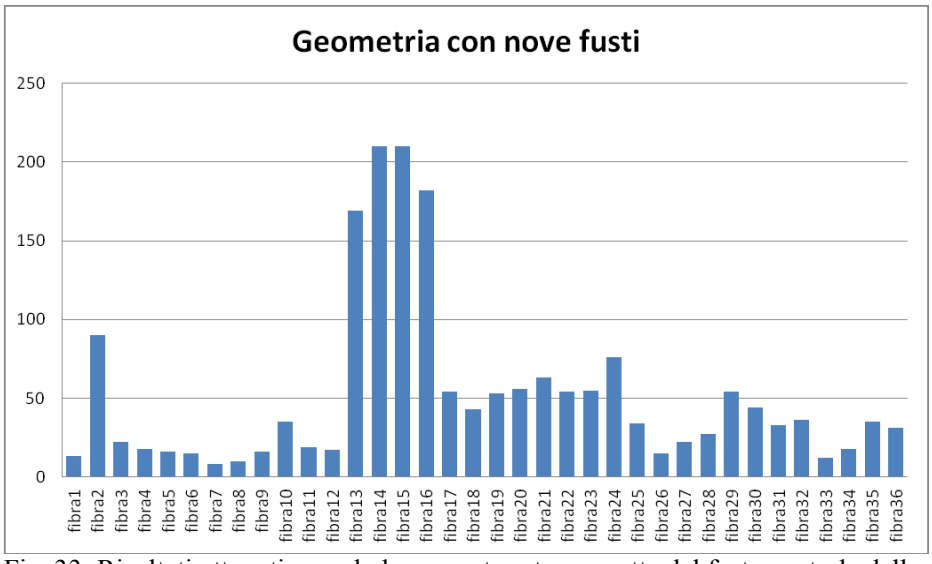


Fig. 33: Risultati ottenuti quando la sorgente estesa emette dal fusto centrale della pila di destra.

Come si osserva dal grafico, le fibre del fusto danneggiato contano un numero di eventi maggiore.

Conclusioni

Con questo lavoro svolto presso i Laboratori Nazionali del Sud dell'INFN si è cercato di sviluppare un nuovo dispositivo per il monitoraggio delle scorie radioattive contenute nei fusti di un sito di stoccaggio. In particolare si è cercato di simulare la presenza di fibre scintillanti attorno ad un fusto danneggiato, da cui fuoriesce materiale radioattivo, al fine di valutarne la risposta in termini di efficienza di rivelazione.

Tale valutazione condotta per mezzo del simulatore Geant4 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, che è stato descritto nel capitolo 1.

Dai risultati ottenuti con le diverse simulazioni possiamo concludere che l'esito delle simulazioni è in accordo con quanto ci si aspetterebbe dal comportamento reale dei materiali.

Si potrebbe, quindi, simulare l'intero sistema di monitoraggio simulando geometrie più vaste e simulando un qualsiasi numero di fibre attorno a ciascun fusto, disponendo tali fibre sia in verticale che in orizzontale in modo da creare una griglia, per ottenere informazioni più precise. Ciò è possibile in quanto questo tipo di rivelatore permette di contare il numero di eventi e quindi il loro rate; questo permette di stabilire una eventuale rottura del fusto, poiché in conseguenza di tale rottura ci sarebbe un aumento del rate di eventi rivelati. Con questo sistema di rivelazione è possibile individuare oltre che il fusto danneggiato, all'interno del sito di stoccaggio, anche il punto di fuoriuscita del materiale radioattivo del fusto stesso riducendo cosi il tempo di intervento e quindi la probabilità di incidenti.

Bibliografia

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

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

[3] Pappalardo, Barbagallo, Bellini, Capogni, Cosentino, Febbraro, Finocchiaro, Greco, Scirè, Scirè, An Online Monitoring System for Nuclear Waste Storage 1st International Conference on Advancements in Nuclear Instrumentation, Measurement Methods and their Applications, Marseille (France), June 2009

http://www.animma.com

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

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

[6] Cosentino, Barbagallo, Capogni, Febbraro, Finocchiaro, Greco, Pappalardo, Scirè, Scirè, A detector mesh for online monitoring of nuclear waste storage 14th International Conference on Emerging Nuclear Energy Systems, Ericeira (Portugal), July 2009

http://www.icenes2009.itn.pt

[7] Geant4 Users' Documents Physics Reference Manual

http://geant4.web.cern.ch/geant4/G4UsersDocuments/UsersGuides/PhysicsReference Manual/html/ PhysicsReferenceManual.html

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

[9]W.R. Leo, Techniques for physics experiments, Lausanne 1987

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

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

[12] Knoll G.F., Radiation detection and measurement, John Wiley Sons Inc., New York 1989.

[12] Scintillation product – Scintillating Optical Fibers, SaintGobain Crystals

[13] B.H. Hasegawa, The physics of Medical X-Ray Imaging, Medical Physics Publishing Company, USA 1991

[14] Particle Gun & GPS

http://hypernews.slac.stanford.edu/HyperNews/geant4/get/eventtrackmanage/655.html

[15] Geant4 User's Guide for Application Developer

http://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/ForApplicationDeveloper/html/inde x.html

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

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

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

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

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

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

[22]J. Allison et al.: Geant4 developments and applications, IEEE Transactions on Nuclear Science 53 No. 1 (2006) 270278

Ringraziamenti

A conclusione di questo lavoro vorrei ringraziare tutti coloro che mi sono stati accanto lungo questo mio percorso.

Ringrazio il prof. Malgeri e il prof. Finocchiaro per la loro disponibilità e la loro pazienza.

Ringrazio l'Ing. Vecchio per il suo supporto e la pazienza avuta durante il mio lavoro di tirocinio.

Un grazie va alle mie colleghe Maria, Giusy e Eleonora con le quali ho condiviso questi anni di università. Grazie anche a Giusi, Mary e Lory che mi sono state accanto e hanno condiviso con me gioie e dolori.

Ringrazio la mia famiglia e il mio amore Giovanni che mi hanno sostenuto durante questi anni dandomi la forza per raggiungere questo mio obiettivo.

Ringrazio te mia piccola Giulia che mi hai fatto capire che la vita è una lotta continua dal primo giorno in cui si viene messi al mondo e che l'importante è non mollare mai, ma lottare….grazie mia piccola grande "guerriera".