Tecniche di analisi numerica e simulazione

Simulazione di un rivelatore di vertice

Irene Amateis, Federico Bonaldo

November 2022

1 Introduzione

Viene preso in esame un collider e si studia il tipico esperimento di collisione di particelle.
Si considera che l'apparato abbia simmetria cilindrica e sia costituito da una beam pipe di berillio in cui vengono lanciati due fasci di particelle in direzioni opposte, identificate con l'asse Z, ed intorno da un rivelatore di vertice. Tale rivelatore è costituito da due piani di pixel al silicio ed è coassiale con la beam pipe. Dalla collisione dei fasci viene prodotto un certo numero di particelle che dovranno essere intercettate dal rivelatore. Si considera che i due layer del rivelatore siano lunghi $27cm$ e di spessore $0,2mm$ e che il raggio della sezione sia di 4 e $7cm$ rispettivamente. Si decide di porre l'origine di un sistema di riferimento cartesiano al centro di questi cilindri. La beam pipe invece viene assunta infinitamente lunga, con sezione di raggio $3cm$ e spessore $0,8mm$.

Lo scopo del programma è simulare tale apparato. Esso deve essere in grado di ricostruire la coordinata Z del vertice in cui sono generate le particelle. Si definisce quindi il modo in cui si desidera che l'apparato effettui la ricostruzione, in seguito si svolge un'analisi dei risultati per ottenere una stima dell'efficienza e della risoluzione dello strumento. La simulazione usata per questo studio è di tipo FAST2. Il programma ha quindi tre fasi: simulazione, ricostruzione e analisi. Sono descritte più nel dettaglio in seguito.

2 Simulazione

La simulazione viene svolta dalla macro MySimulation.cpp che contiene la funzione principale chiamata MySimulation i cui argomenti sono: il numero di esperimenti simulati (ovvero il numero di vertici generati), un booleano per imporre o meno la presenza di multiple scattering e il seed per l'estrazione di numeri casuali.

In questa macro viene estratta la posizione del vertice primario di interazione e il numero di particelle che vengono prodotte da tale interazione, ovvero la molteplicità. Per ogni particella viene anche generata la sua direzione di moto, data estraendo due angoli indicati con θ e ϕ, rispettivamente angolo azimutale e zenitale. Vengono poi calcolate le intersezioni di queste particelle nei due layer del rivelatore, verificando che i risultati appartengano effettivamente alla lunghezza dei rivelatori.
Infine, i risultati vengono salvati in un tree con tre branch. Nel primo vengono salvati, per ogni evento, le coordinate del vertice e la molteplicità. Negli altri due vengono salvate le intersezioni con i due layer rispettivamente, identificate da oggetti della classe custom MyIndex.

La macro utilizza dei metodi che sono stati definiti all'interno di classi custom.

2.1 MyGen

La classe MyGen eredita da TRandom3 e implementa tutti i metodi necessari alla simulazione per la generazione di numeri pseudocasuali.

- **Generazione delle coordinate del vertice primario**: Distribuzione normale centrata in zero e con dispersione $0,1mm$ per le coordinate X, GenX(), e Y, GenY(); Per la coordinata Z una distribuzione normale centrata in zero e con dispersione 5,3 cm, GenZ(), e una distribuzione uniforme su $[-30; 30]cm$, GenZUni()

- **Generazione della molteplicità**: Distribuzione fissa a 50 particelle, GenMultFixed(); Distribuzione uniforme tra 1 e 81 particelle, GenMultUni(); Distribuzione assegnata realizzata da GenMultDistro() e da MyOpenFile() che apre un file .root su cui è salvata la distribuzione per l'estrazione della molteplicità
- **Generazione della direzione delle particelle**: Distribuzione uniforme tra 0 e 2π per ϕ, GenPhi(); Distribuzione assegnata per θ iniziale, GenTheta(), ottenuta a partire da una distribuzione assegnata di pseudorapidità a cui si accede tramite MyOpenFile(); Distribuzione normale con dispersione $\sqrt{2}mrad$ per l'angolo θ', GenThetap(), usato nel multiple scattering
- **Generazione dei punti di noise**: GenNoise() prende in argomento un array la cui prima componente è una coordinata Z, estratta uniformemente su $[-13,5; 13,5]cm$, e la seconda è un angolo ϕ, estratto uniformemente su $[0; 2\pi]$.

2.2 MyInt

La classe MyInt, che eredita da TObject, viene utilizzata per calcolare i punti di intersezione delle particelle generate in ogni evento con la beam pipe e i due layer.

- **Impostare i valori iniziali dei data member**: SetTheta e SetPhi usate per impostare i valori degli angoli; StartPoint per le coordinate X, Y e Z in base all'argomento dei metodi rispettivamente
- **Calcolo del'intersezione traccia-rivelatore/beam pipe**: svolto tramite i metodi Calc1, Calc2 e Calc3 i quali, considerando la direzione della singola particella scritta come una retta in forma parametrica, calcolano i coefficienti
- **tInt(double x, double y, double z, double f, double t, double R)** calcola l'intersezione della retta con i diversi layer e beam pipe nel caso di assenza di multiple scatterig oppure **tInt(double x, double y, double z, double* u, double R)** se è stata imposta la condizione di multiple scattering
- **SetX, SetY, SetZ**: chiamano al loro interno i precedenti metodi privati e riscrivono il valore dei data member come risultato delle intersezioni
- **CheckZ()**: verifica se le particelle hanno effettivamente intersecato i layer
- **MyClear()**: ripulisce i valori dei data member impostandoli a zero

2.3 MyScatter

La classe MyScatter, che eredita da TObject, implementa il multiple scattering.

- **StartScatter()**: impone ai data member i valori inziali la prima volta che viene effettuato multiple scattering
- **SetThetap, SetPhip**: impongono i valori degli angoli θ e ϕ dopo lo scattering per il sistema di riferimento $(O'x'y'z')$, essi sono generati casualmente nella macro di simulazione
- **rotate(double *u)**: calcola la nuova direzione della particella salvandola come elementi dell'array che viene passato per argomento
- **SetPhi, SetTheta**: salvano i nuovi angoli della particella nel sistema di riferimento del laboratorio a partire dal calcolo precedente
- **ClearScatter()**: ripulisce i valori dei data member impostandoli a zero

2.4 MyIndex

La classe MyIndex, che eredita da TObject, ha come data member la coordinata Z dell'intersezione della singola particella con uno dei due layer, l'angolo ϕ e un indice che indentifica tale particella nella molteplicità. Queste informazioni, tramite il costruttore standard, verranno salvate nel tree di output per ogni particella generata e per ogni intersezione con i rivelatori.
Il metodo **SetData(double z, double phi, int index)** viene usato nella simulazione per impostare i valori dei data member.
Il metodo **SetData(double z, double phi)** verrà usato per lo stesso scopo nella fase di ricostruzione.

3 Ricostruzione

La ricostruzione viene effettuata tramite la macro MyReconstruction.cpp, essa contiene la funzione MyReconstruction(unsigned int seed) il cui argomento è il seed usato per l'estrazione di numeri pseudocasuali.
Si comincia effettuando lo smearing gaussiano dei punti di intersezione delle particelle con i layer.
A partire da questi nuovi punti di impatto si costruiscono per ogni evento delle tracklet: si accoppia un punto sul layer1 con uno sul layer2; la retta passante per essi potrà intersecare l'asse Z individuando così una possibile posizione di vertice primario.
La funzione TrackZ(double z1, double R1, double z2, double R2) restituisce il valore di tale coordinata.
I punti vengono accoppiati se la differenza in angolo delle direzioni delle loro traiettorie (quantificate da *phi*) è inferiore a un angolo massimo posto pari a $6mrad$.
Le coordinate Z così calcolate vengono inserite in un vector che viene usato per riempire un istogramma.
Per il calcolo della coordinata Z ricostruita del vertice, Zrec, si considerano più casi:

- Se il vector è vuoto: Il programma non ricostruisce e Zrec è posto pari a $50cm$.
- Se il vector contiene un solo elemento: Zrec è pari a quell'unico elemento.
- Se il vector ha una dimensione superiore a 1: Si cercano i bin che contengono il numero massimo di elementi al loro interno e si crea un vector contente gli indici di tali bin. Per ognuno di essi si considera un "gruppo" di bin costruito prendendo il bin di riferimento e tutti i bin adiacenti fino a quando non si trova un "buco" (ovvero una cella vuota), a questo punto si calcola una media dei centroidi dei bin appartenenti a tale gruppo pesati sul numero di ingressi del singolo bin e si conta il numero totale di entries (denominatore di tale media). Si presentano diversi casi:
  1. C'è un solo gruppo con numero di ingressi totale superiore agli altri: Zrec è calcolato come la media degli elementi del vector che distano al massimo $1,5mm$ dalla media pesata dei centroidi del gruppo.
  2. Ci sono più gruppi con numero di ingressi totali massimo ma stesso valore di media dei centroidi: essi sono lo stesso gruppo e Zrec è la media degli elementi che distano al massimo $1,5mm$ dalla media pesata dei centroidi.
  3. Ci sono più gruppi con numero di ingressi totali massimo ma diverso valore di media dei centroidi: si tratta di gruppi equipollenti separati da bin vuoti, non si ricostruisce e Zrec è posto pari a $50cm$.

Infine, si salva su un file .root i risultati tramite un tree costituito da tre branch contenenti per ogni evento la molteplicità di particelle generate, la coordinata Z simulata del vertice e la coordinata Z ricostruita.

Le classi utilizzate per la ricostruzione sono MyIndex, descritta precedentemente, e MyRec.

3.1 MyRec

I metodi **GenZr()** e **GenPhir()** vengono usati per lo smearing gaussiano dei punti di intersezione su i layer: estraggono numeri da gaussiane centrate in zero i quali vengo sommati, nella macro di ricostruzione, a Z e ϕ di ogni punto di intersezione calcolato nella simulazione.
Il metodo **TrackZ(double z1, double R1, double z2, double R2)** restituisce la coordinata Z dell'intersezione della retta di tracklet con l'asse Z.

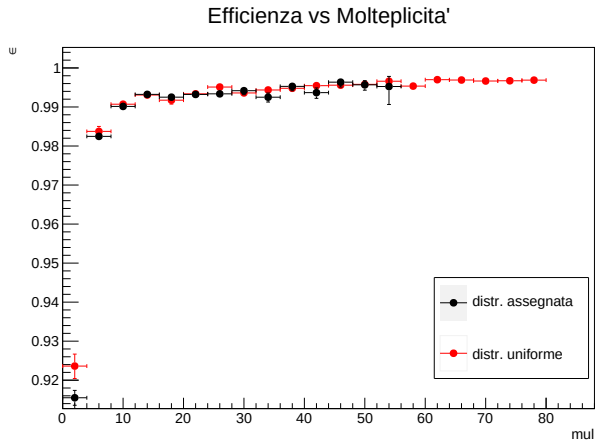
4 Analisi

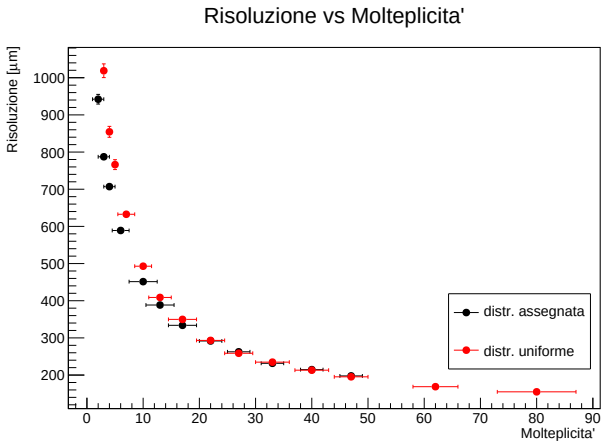
Viene effettuata un'analisi dei risultati ottenuti per valutare le prestazioni dell'apparato simulato. Le macro utilizzate per la realizzazione dei grafici sono le seguenti:

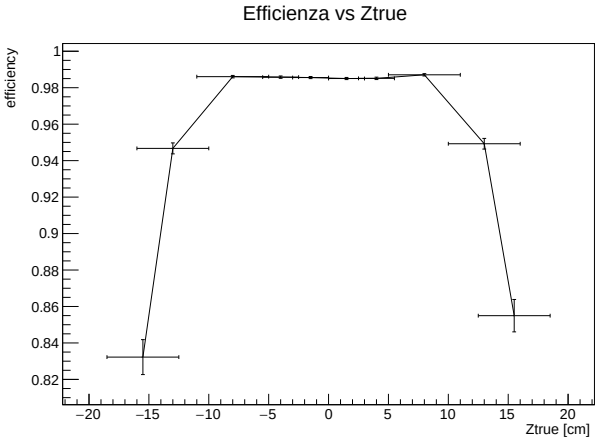
- MyReconstruction.cpp per i grafici di efficienza vs molteplicità tramite la classe root TEfficiency
- analisiMultUni.cpp per il grafico di risoluzione vs molteplicità nel caso di molteplicità estratta uniformemente
- analisiZUni.cpp per il grafico di efficienza vs Zsim (coordinata Z del vertice simulata) e risoluzione vs Zsim nel caso di molteplicità fissata a 50 e Zsim estratto uniformemente

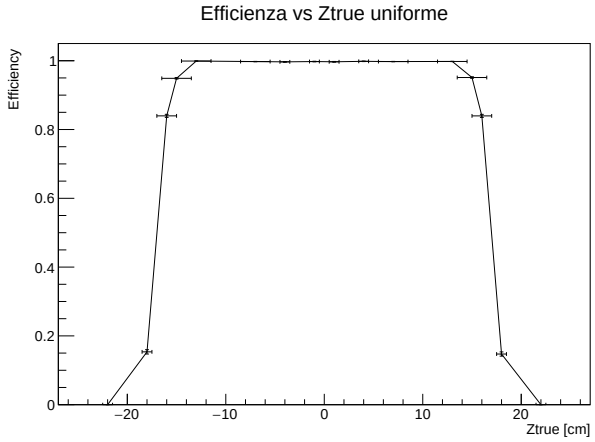
- analisiDistro.cpp per i grafici di risoluzione vs molteplicità, risoluzione vs Zsim, efficienza vs Zsim nel caso di molteplicità estratta da distribuzione assegnata e Z gaussiano
- readGraph.cpp e MyGraph.cpp servono per sovrapporre grafici realizzati da macro diverse su uno stesso canvas

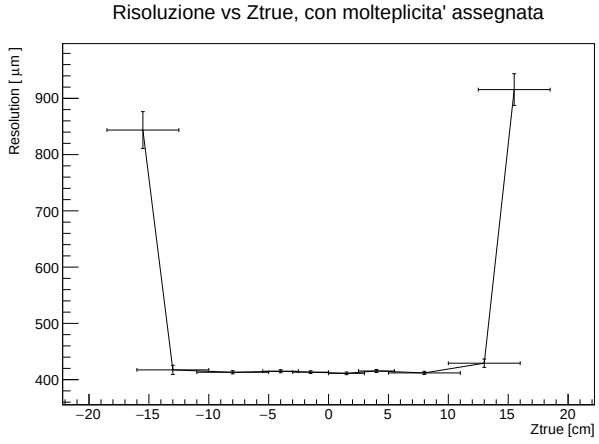
I grafici riportati sono stati realizzati generando 200000 eventi, 20 punti di noise per ogni layer e in presenza di multiple scattering.

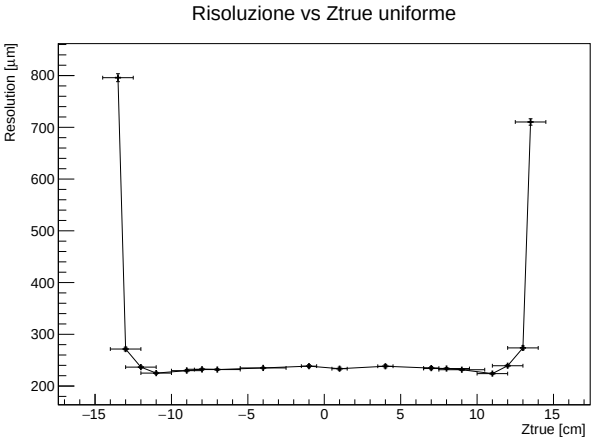












5 Esecuzione del Programma

Le macro di simulazione e ricostruzione si trovano in due cartelle separate dentro le quale è possibile eseguire le due fasi del programma.

Dalla cartella *simulazione*:

1. .x compilemyclass.C
2. MySimulation();

Dalla cartella *ricostruzione*:

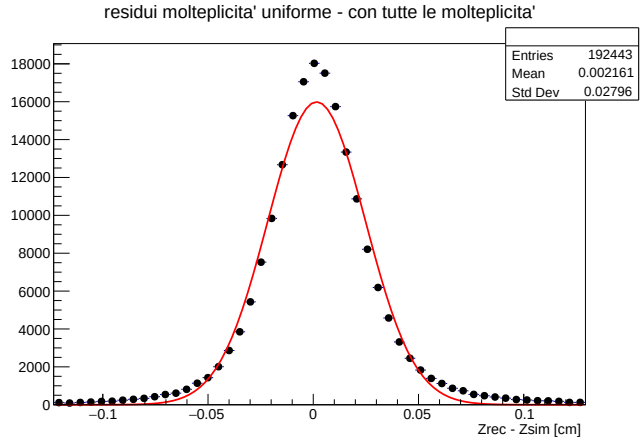
1. .x compilemyclass.C
2. MyReconstruction();

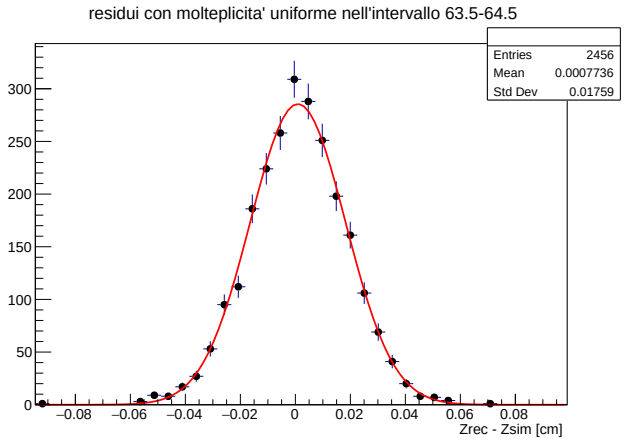
Per analizzare la ricostruzione, ossia valutare il rivelatore in esame, bisogna compilare e lanciare le diverse macro sopra elencate, a titolo di esempio:
.x analisiMultUni.cpp

6 Risultati

In questa sezione si riportano i risultati ottenuti dalla ricostruzione. Ci si domanda quanto bene il rivelatore sia stato in grado di ricostruire le posizioni dei vertici lungo l'asse Z. Per farlo si costruisce una distribuzione di residui, definiti come $Z_{sim} - Z_{rec}$.

La simulazione presa in considerazione è stata eseguita con una generazione di 200000 vertici, con distribuzione di molteplicità uniforme nell'intervallo $1 - 81$, con Zsim da distribuzione normale e con 20 punti di noise per ogni layer.
I grafici sono realizzati tramite la macro residui.cpp (per lanciarla eseguire il comando .x residui.cpp nella cartella *analisi*).





La distribuzione dei residui per tutte le molteplicità non segue una distribuzione gaussiana.
A fianco, è rappresentato il grafico dei residui con molteplicità selezionata a 64, in questo caso il fit segue correttamente una distribuzione normale.
Si ottiene dal fit un rapporto $\chi^2/N = 17.24/22$ con $P_{value} = 0.750$.

7 Riferimenti bibliografici

1. G. Cowan, Statistical Data Analysis