Tecniche di Analisi Numerica e Simulazione

-

Progetto di Esame

Rares Sorin Drosu (rares.drosu@edu.unito.it)
Lorenzo Visca (lorenzo.visca846@edu.unito.it)

A.A. 2025/2026

DIRECTORY DEL PROGETTO

- Il progetto è scaricabile al link: https://github.com/lorenzovisca846/Project2026TANS

COMPILAZIONE

- Comandi per la compilazione (da eseguire nella home directory del progetto):

```
mkdir outputs
mkdir outputs/plots
mkdir build
cd build
cmake ..
make
```

ESECUZIONE

- Una volta compilato il progetto si possono avviare sequenzialmente gli eseguibili. I comandi da eseguire nella directory `build` sono:

```
./simulation
./reconstruction
./analysis
```

- In questo modo, nella directory `outputs/plots` vengono salvati tutti i plot.

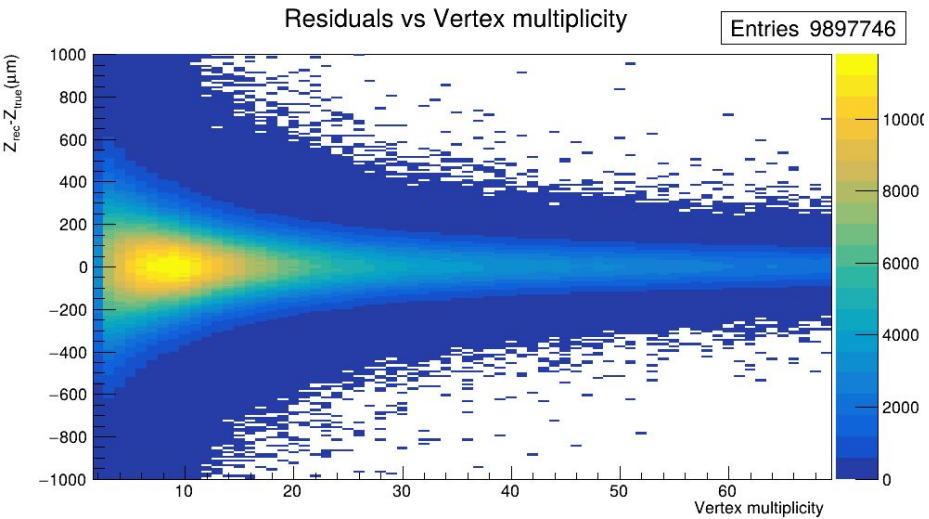
RISULTATI

- Nelle slides successive vengono mostrati i risultati di due run con 10’000’000 eventi ciascuna: in una run la coordinata Z del vertice viene estratta da una distribuzione gaussiana con $\sigma = 5.3$ cm, nell’altra l’estrazione viene effettuata da una distribuzione uniforme (-25 cm, 25 cm).

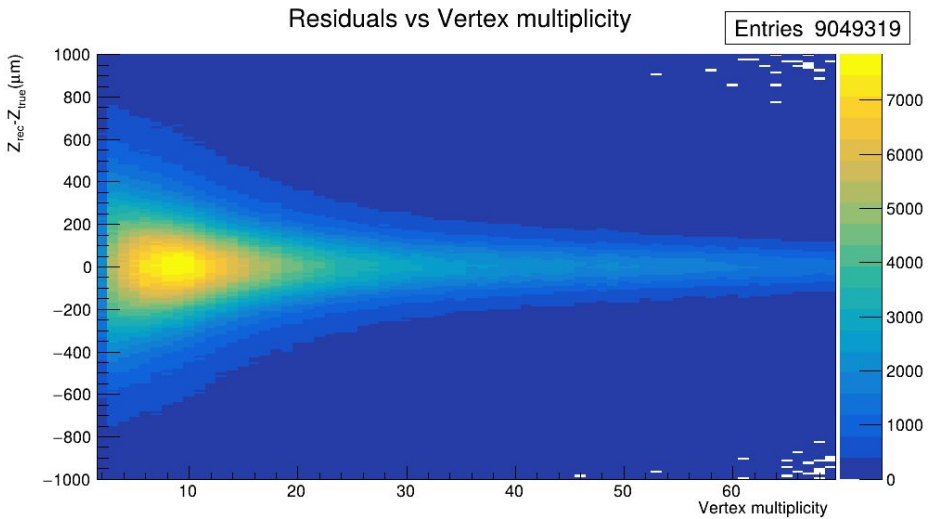
RISULTATI

- Distribuzione dei residui in funzione della molteplicità del vertice.

Distribuzione gaussiana



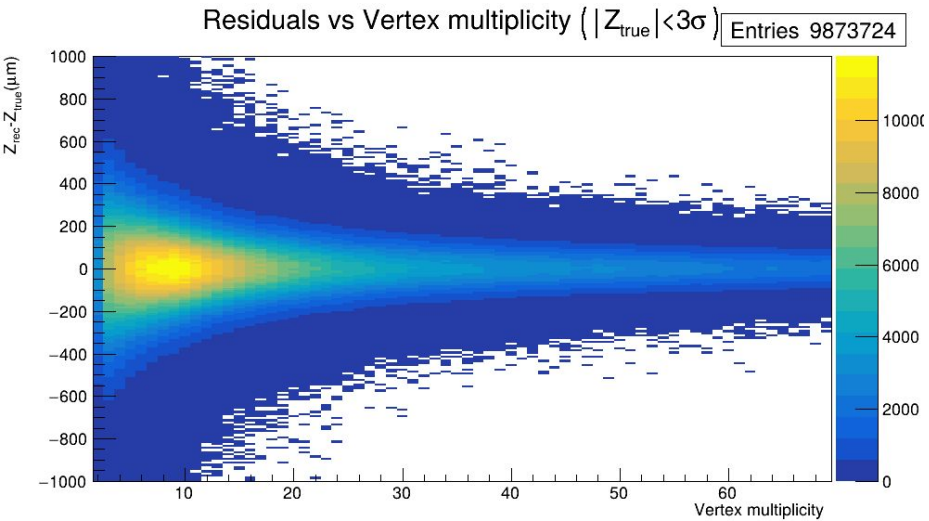
Distribuzione uniforme



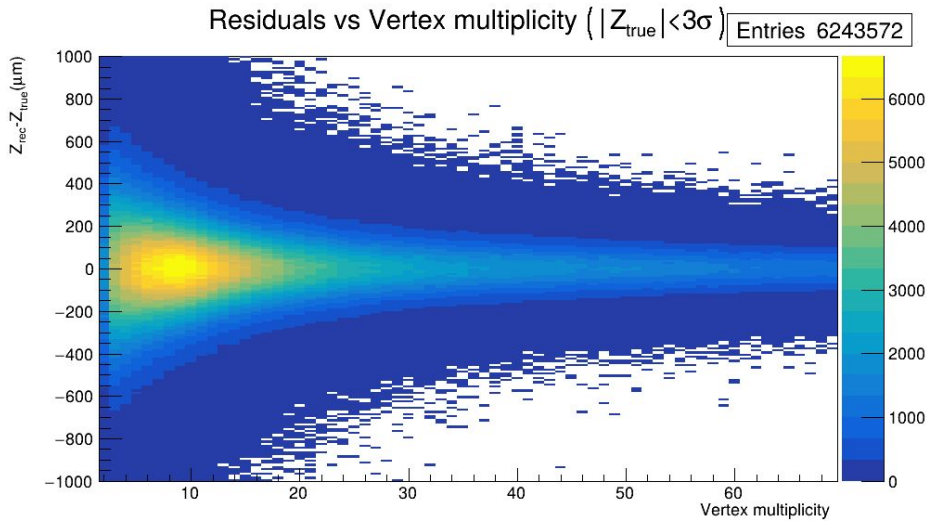
RISULTATI

- Distribuzione dei residui in funzione della molteplicità del vertice: taglio su Z del vertice entro 3σ dal centro.

Distribuzione gaussiana



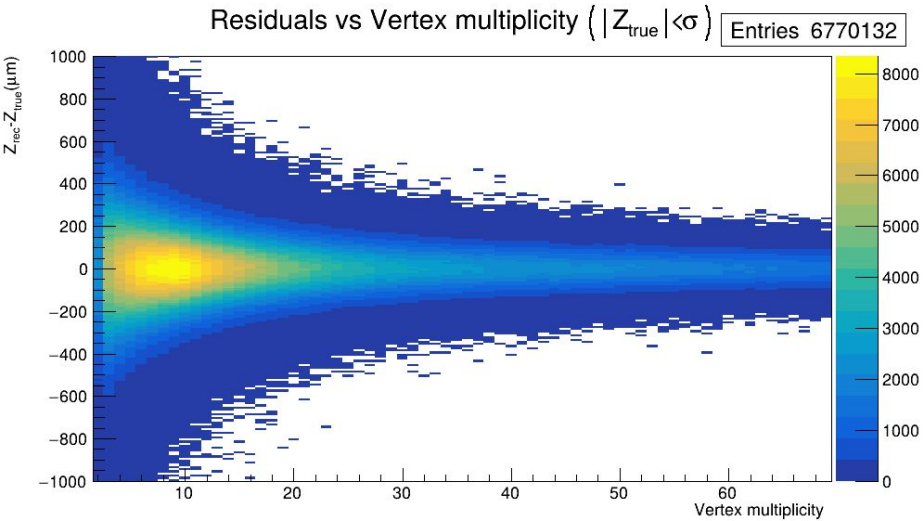
Distribuzione uniforme



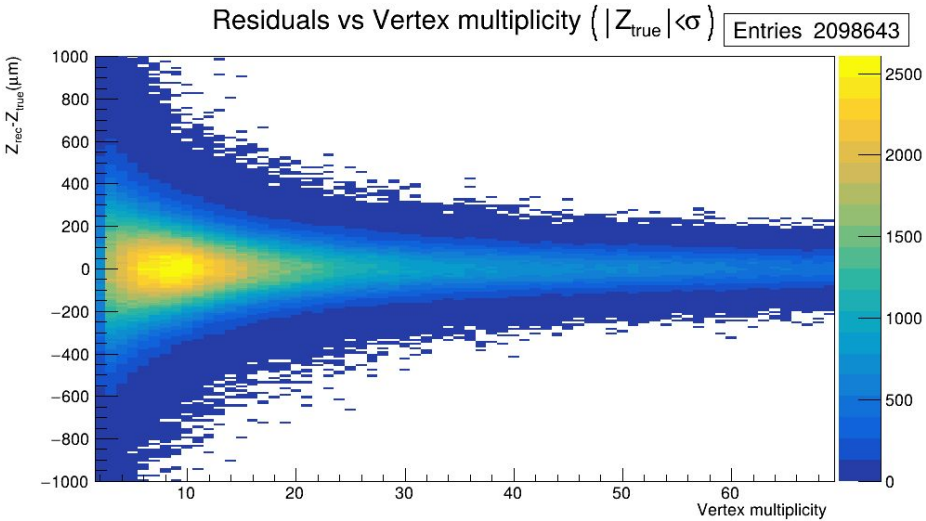
RISULTATI

- Distribuzione dei residui in funzione della molteplicità del vertice: taglio su Z del vertice entro 1σ dal centro.

Distribuzione gaussiana



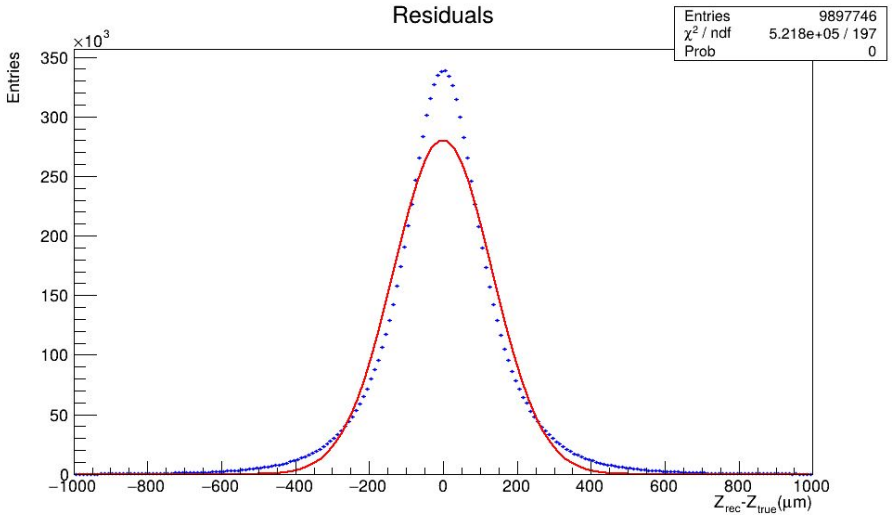
Distribuzione uniforme



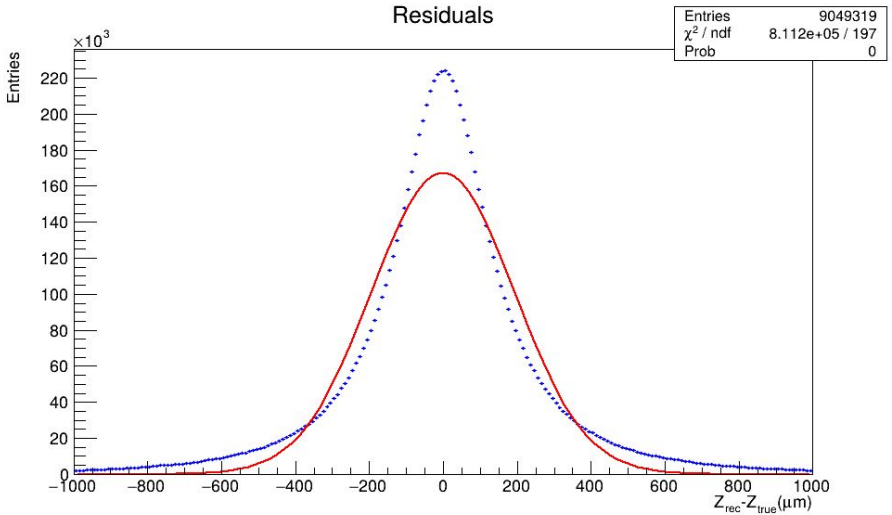
RISULTATI

- Distribuzione dei residui: dal tentativo di fit si nota che l'andamento è marcatamente non gaussiano.

Distribuzione gaussiana



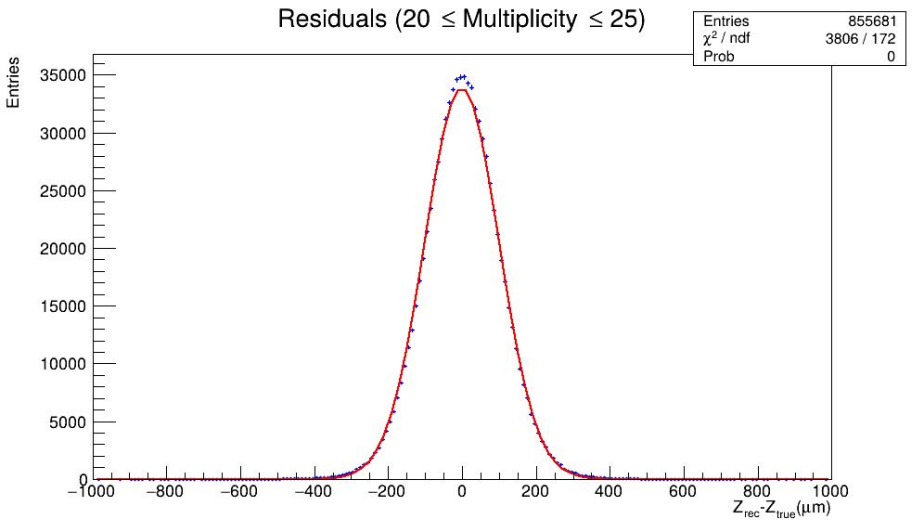
Distribuzione uniforme



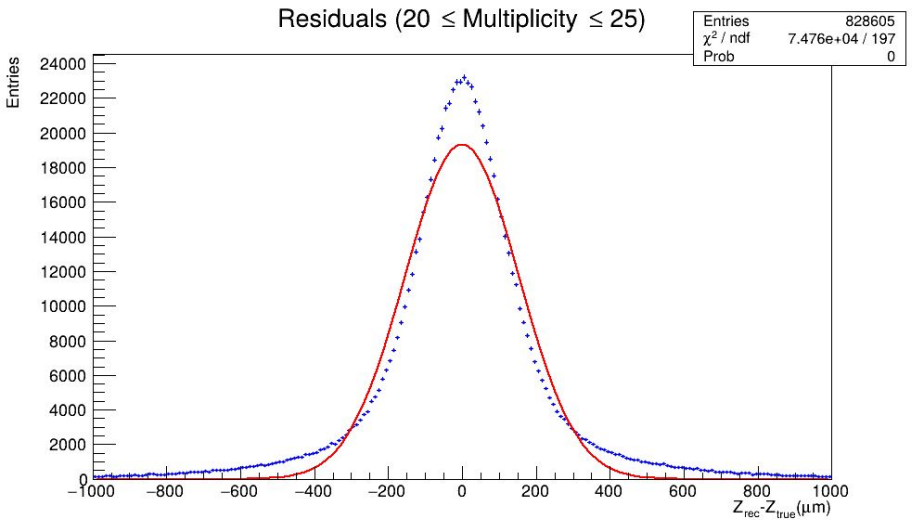
RISULTATI

- Distribuzione dei residui in un intervallo ristretto di molteplicità.

Distribuzione gaussiana



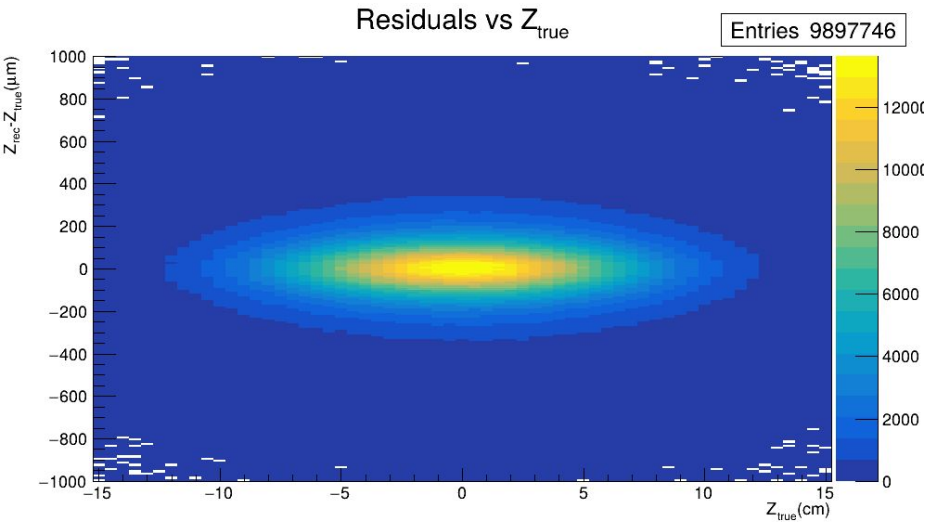
Distribuzione uniforme



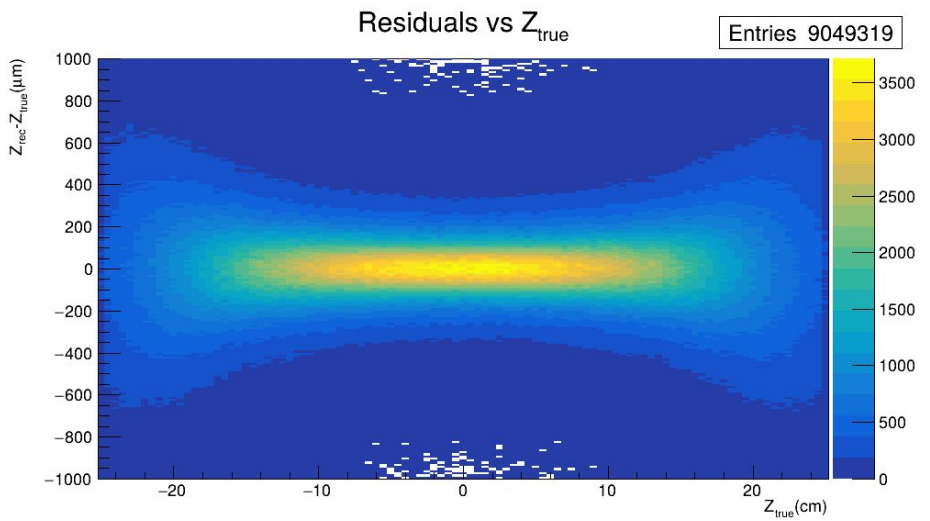
RISULTATI

- Distribuzione dei residui in funzione della coordinata Z del vertice.

Distribuzione gaussiana



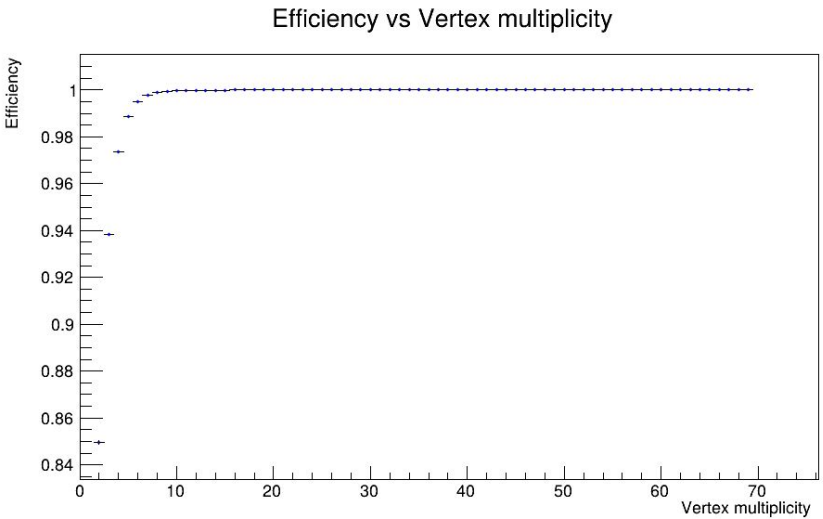
Distribuzione uniforme



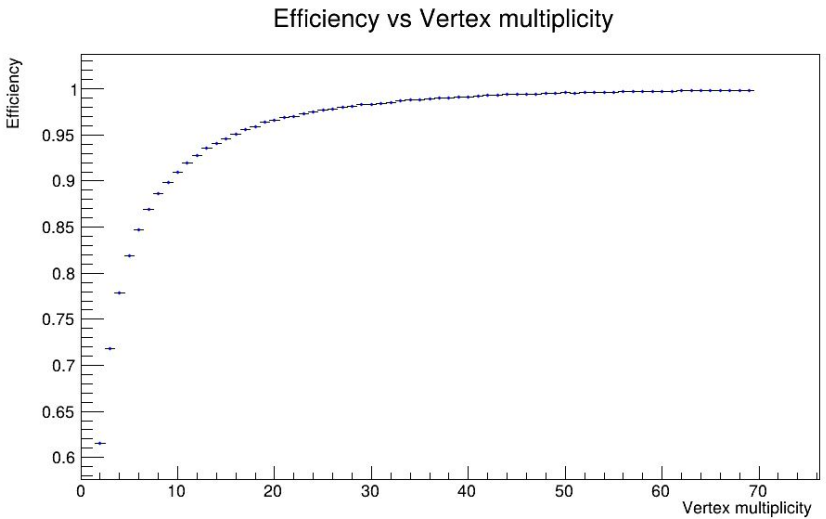
RISULTATI

- Efficienza in funzione della molteplicità.

Distribuzione gaussiana



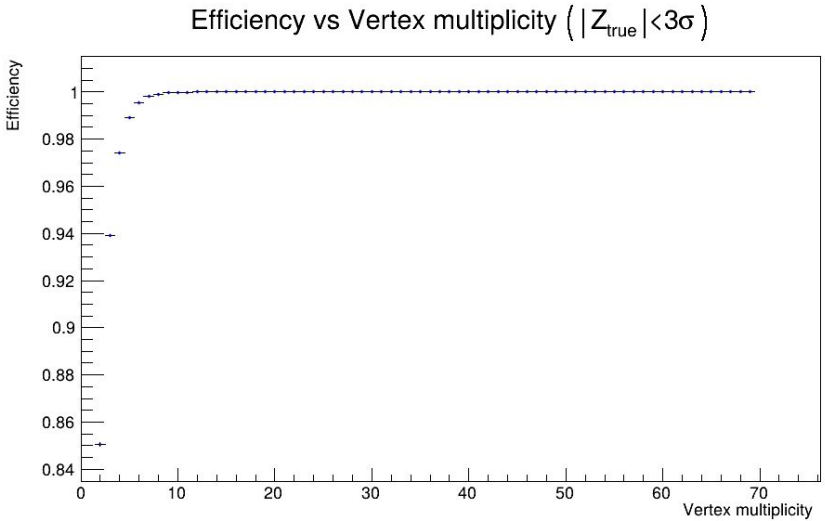
Distribuzione uniforme



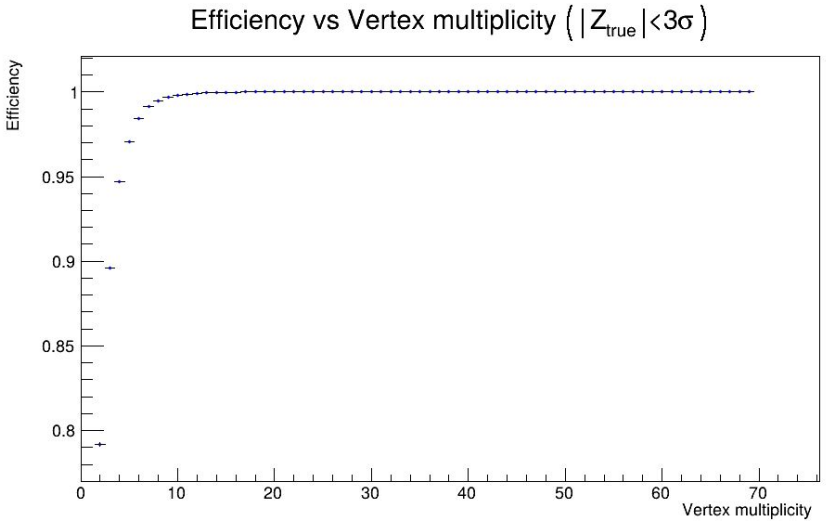
RISULTATI

- Efficienza in funzione della molteplicità: taglio su Z del vertice entro 3σ dal centro.

Distribuzione gaussiana



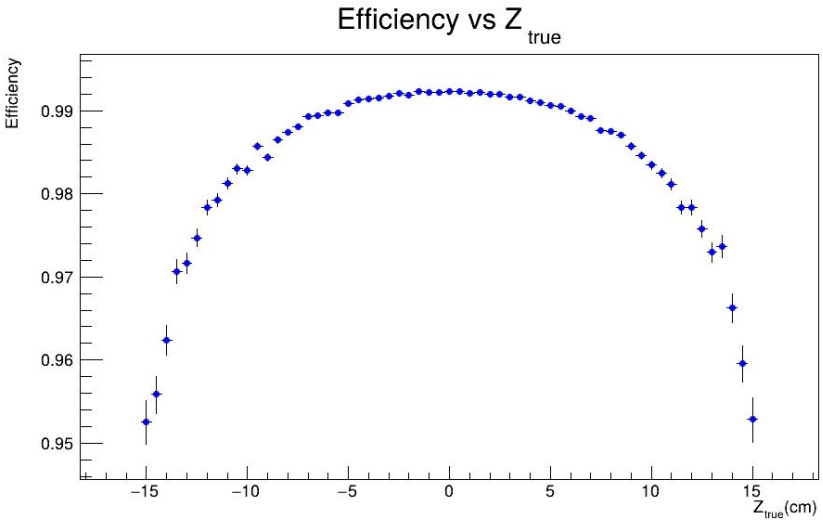
Distribuzione uniforme



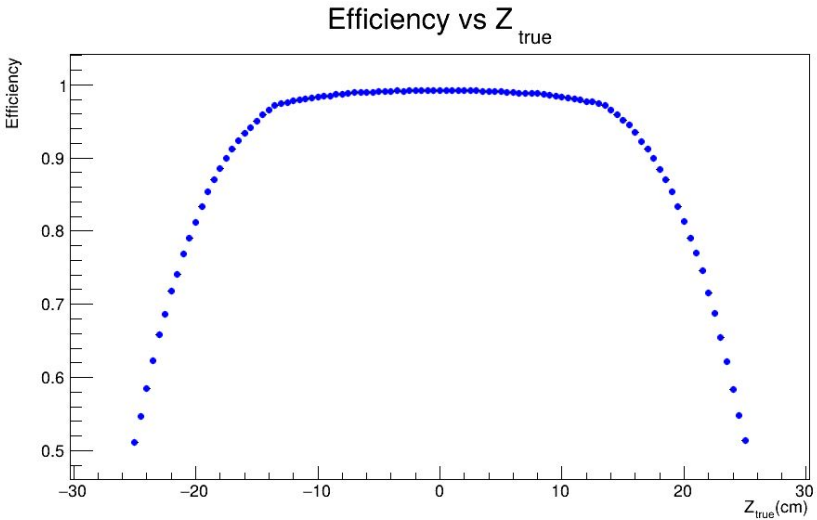
RISULTATI

- Efficienza in funzione della coordinata Z del vertice.

Distribuzione gaussiana



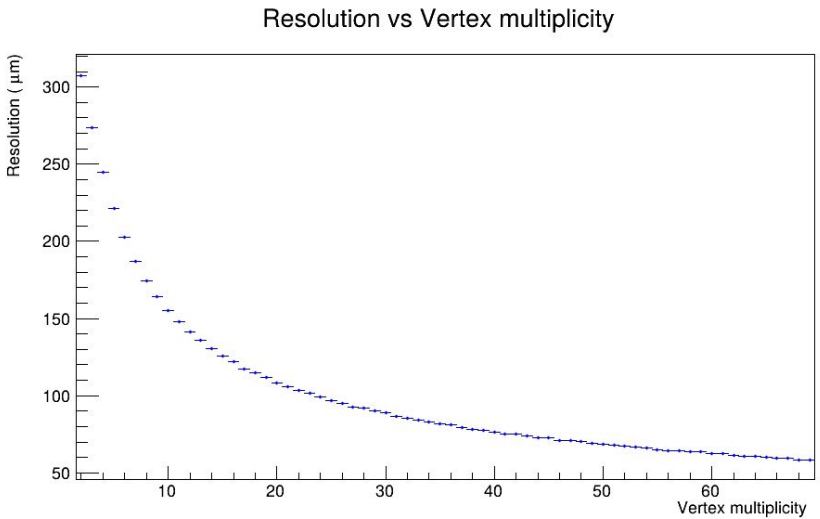
Distribuzione uniforme



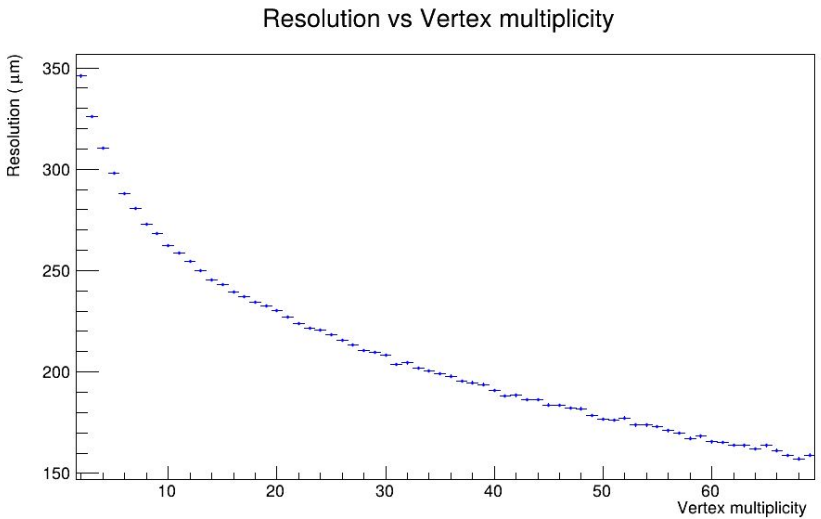
RISULTATI

- Risoluzione in funzione della molteplicità.

Distribuzione gaussiana



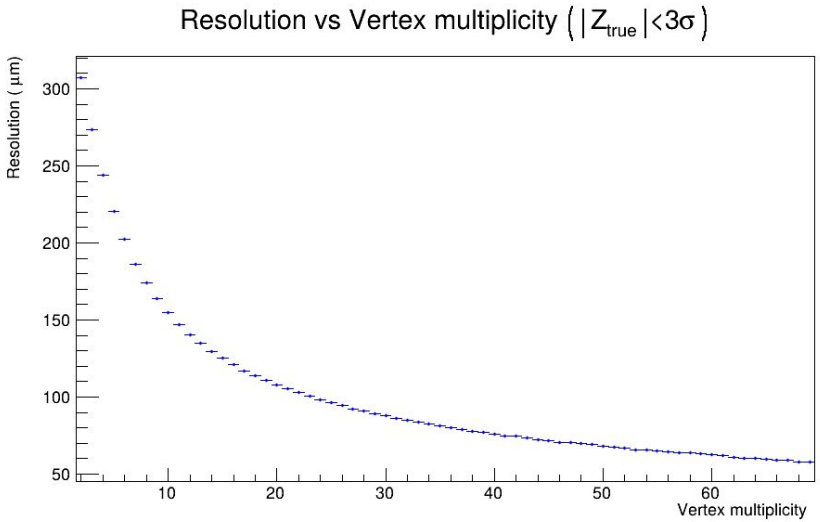
Distribuzione uniforme



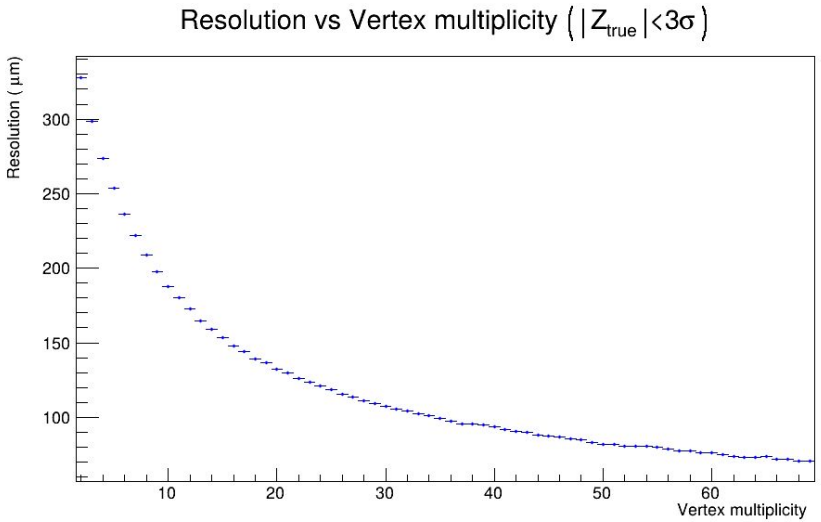
RISULTATI

- Risoluzione in funzione della molteplicità: taglio su Z del vertice entro 3σ dal centro.

Distribuzione gaussiana



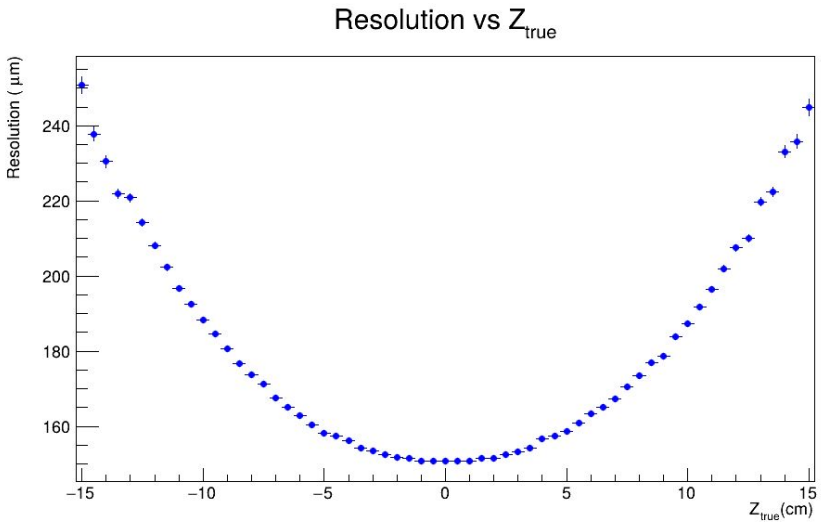
Distribuzione uniforme



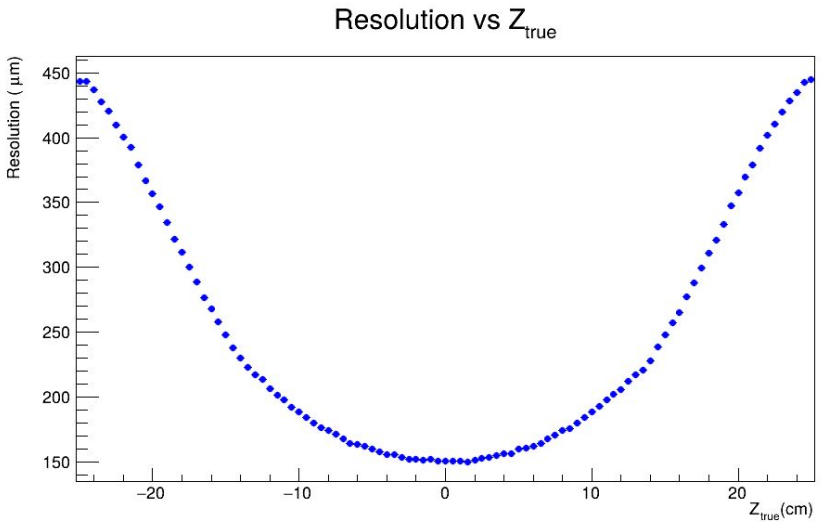
RISULTATI

- Risoluzione in funzione della coordinata Z del vertice.

Distribuzione gaussiana



Distribuzione uniforme



EVENT DISPLAY

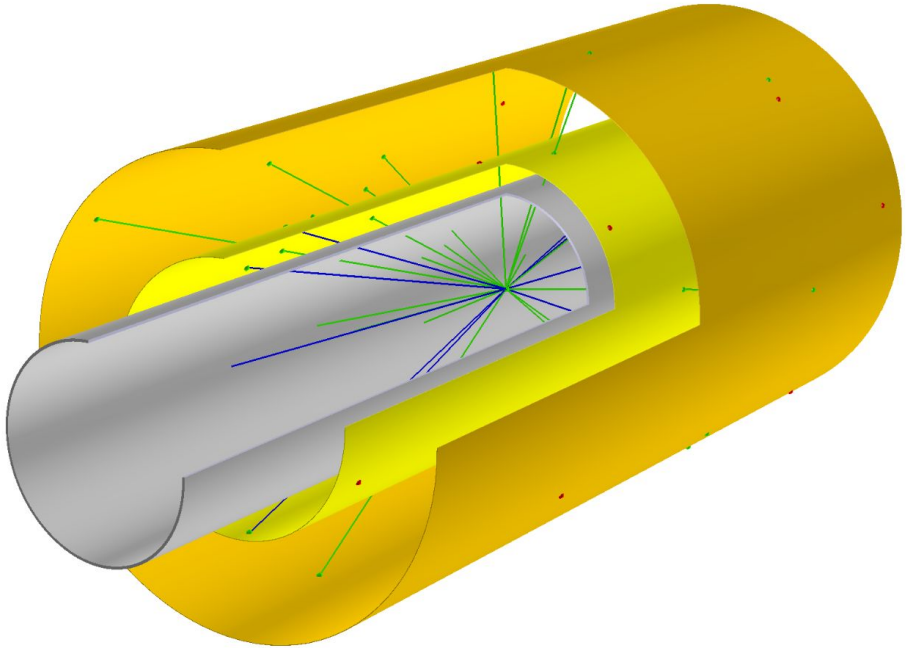
- Impostando la variabile `DISPLAY` su `true` in `main_simulation.cpp` viene eseguita una simulazione con un solo evento utilizzando file di input e output prefissati, utilizzabili per l'event display.

- Per visualizzare la simulazione, utilizzare il seguente comando nella home directory del progetto:

```
root eventdisplay.C
```

EVENT DISPLAY

- Esempio di evento: i punti rossi sul rivelatore sono noise, le tracce verdi hanno generato hits su entrambi i rivelatori, le tracce blu sono fuori dall'accettanza geometrica di almeno un layer.



FILE DI CONFIGURAZIONE

- Quando si eseguono i programmi i valori delle variabili vengono impostati utilizzando la classe ROOT `TEnv`, che permette di leggere i valori da file.

- Se non viene specificato diversamente, il file utilizzato è `fullConfig.txt` situato nella directory `config`.

- È possibile utilizzare un file di configurazione personalizzato passandone il nome come argomento ai programmi (specificandone il path rispetto alla directory `config`), ad esempio:

```
./simulation customConfig.txt
```

FILE DI CONFIGURAZIONE

- Di seguito viene presentata una panoramica dei principali valori la cui modifica può risultare di interesse:

- `Events`: numero di eventi
- `MultScattering`: valore booleano che determina se considerare lo scattering multiplo
- `Zedges_uniform`: se la coordinata Z del vertice segue una distribuzione uniforme, rappresenta la semiampiezza dell'intervallo
- `Minimum` / `Maximum`: valori limite della molteplicità
- `NoiseEnabled`: valore booleano che determina se inserire hits di rumore
- `NoiseRate`: rate medio delle hits di rumore su ciascun layer
- `RunningWindowSize`: ampiezza della running window in ricostruzione
- `Residuals_zlim`: valore a cui vengono tagliate le code nei plot dei residui
- `MultZoom_Min` / `MultZoom_Max`: intervallo di molteplicità su cui calcolare i residui

FILE DI CONFIGURAZIONE

- `Generation`: stringa di tre caratteri per scegliere le funzioni di generazione dei valori casuali.
  - Primo carattere (generazione del vertice):
    - `g`: distribuzione gaussiana in x,y,z
    - `u`: distribuzione gaussiana in x,y, distribuzione uniforme in z
    - `o`: vertice nell'origine
  - Secondo carattere (generazione della molteplicità):
    - `h`: estrazione da istogramma
    - `u`: distribuzione uniforme tra `Minimum` e `Maximum`
    - `f`: valore fissato a `Maximum`

FILE DI CONFIGURAZIONE

- `Generation`: stringa di tre caratteri per scegliere le funzioni di generazione dei valori casuali.
  - Terzo carattere (generazione del rumore):
    - `p`: distribuzione Poissoniana con media `NoiseRate` troncata a `MaxNoise`
    - `u`: distribuzione uniforme su (0,`MaxNoise`)
    - `f`: valore fissato a `NoiseRate` convertito a intero

SIMULAZIONE
(codici)

SIMULAZIONE: CYLINDER

- La classe `Cylinder` descrive la geometria della beam pipe e dei due layer di rivelazione.

- Gli oggetti `Cylinder` vengono costruiti con 5 input:

  - raggio interno `double`
  - lunghezza `double`
  - spessore `double`
  - materiale `string`
  - layer `int`

- I data members vengono inizializzati di conseguenza, gli input "`Be`" e "`Si`" sul materiale definiscono la lunghezza di radiazione corrispondente.

SIMULAZIONE: MYPOINT

- La classe `MyPoint` descrive un punto in coordinate cilindriche, e contiene in memoria la traccia che lo ha generato (utile per l'event display).

- Gli oggetti `MyPoint` vengono costruiti con tre (quattro) input:

  - R `double`
  - φ `double`
  - Z `double`
  - (trackID) (`int`)

- Le funzioni getter consentono anche di ottenere le coordinate cartesiane del punto.

SIMULAZIONE: PARTICLE

- La classe `Particle` contiene le funzioni di trasporto.
- La costruzione di un oggetto `Particle` richiede il puntatore alla classe `SimRandom` che viene utilizzato nel main code.
- La funzione `Init` definisce a partire dal vertice la posizione iniziale della particella, genera i parametri angolari dalle relative distribuzioni e calcola i coseni direttori.
- La funzione `Propagation` prende in input il raggio da raggiungere e trasporta la posizione (x,y,z) a tale raggio.

SIMULAZIONE: PARTICLE (2)

- La funzione `MultScatter` prende in input le caratteristiche dello strato di materiale, calcola lo spessore efficace attraversato e definisce la nuova direzione di propagazione in termini di coseni direttori.

- La funzione `AngleUpdate` definisce i nuovi angoli in coordinate cilindriche in funzione dei coseni direttori.

SIMULAZIONE: SIMRANDOM

- La classe `SimRandom` si occupa della generazione di tutte le variabili casuali. Un oggetto `SimRandom` viene generato a partire da un seed e dagli istogrammi contenenti le distribuzioni assegnate.

- Viene inoltre definito un costruttore che prende anche in input un valore di tipo `double` che definisce la semiampiezza dell'eventuale distribuzione uniforme della coordinata Z del vertice.

SIMULAZIONE: SIMRANDOM (2)

- Le funzioni di generazione sono:
  - `VertGaus`, `VertUnif`, `VertOrig`: funzioni di generazione del vertice
  - `MultUnif`, `MultHisto`, `MultFixed`: funzioni di generazione della molteplicità
  - `NoisePois`, `NoiseUnif`: funzioni di generazione del rumore
  - `PhiDist`: distribuzione uniforme su (0,2π)
  - `ThetaDist`: conversione di una pseudorapidità estratta da istogramma
  - `ScatterDist`: distribuzione dell'angolo polare di multiple scattering
  - `ZDist`: distribuzione uniforme sulla lunghezza dei rivelatori
  - `Smear`: gestisce lo smearing delle hits nella ricostruzione

SIMULAZIONE: MAIN

- Il main code `main_simulation.cpp` definisce i parametri della simulazione ed esegue il loop sugli eventi, creando un `TTree` in output.
- Le funzioni `NoiseU`, `NoiseP`, `NoiseF` generano hits di rumore distribuite uniformemente sulle superfici dei rivelatori utilizzando la relativa distribuzione.
- La funzione `Transport` gestisce il trasporto della particella e la creazione dei punti di hit.
- La funzione `FunctionAssignment` permette di scegliere tra le funzioni di generazione dei valori casuali associando a ogni funzione scelta il puntatore di alias.

SIMULAZIONE: MAIN (2)

- Nel loop vengono eseguite le seguenti operazioni:
  - Generazione del vertice
  - Trasporto alla beam pipe, multiple scattering
  - Trasporto al layer 1, multiple scattering, salvataggio dell'hit
  - Trasporto al layer 2, salvataggio dell'hit
  - Generazione del rumore sui layer
  - Riempimento del `TTree`

- Se la variabile `DISPLAY` è impostata su `true`, al solo scopo di visualizzare l'evento vengono salvate anche le hits sulla beam pipe in un'apposita branch, che normalmente non viene definita.

RICOSTRUZIONE (codici)

RICOSTRUZIONE: CONFIG

- Nella fase di ricostruzione tutte le variabili utilizzate vengono salvate in un oggetto di tipo `Config`, che viene passato alle funzioni di ricostruzione.
- Il costruttore di `Config` utilizza un puntatore a un oggetto della classe ROOT `Tenv` per leggere il file di configurazione.
- Il puntatore utilizzato nella generazione dei valori casuali è univoco ed è salvato all'interno di `Config`.
- La funzione `Print` stampa i valori di configurazione.

RICOSTRUZIONE: TRACKLET

- Un oggetto della classe `Tracklet` definisce la tracklet associata a una coppia di punti sui due layer, di cui viene salvato l'indice.
- Il costruttore di `Tracklet` richiede una coppia di indici relativi alle hits sui due layer.
- La funzione `CalculateTrackletIntersection` prende in input i due punti e ne calcola l'intersezione con l'asse del fascio sul piano R-Z.

RICOSTRUZIONE: MAIN

- Il main code `main_reconstruction.cpp` definisce i parametri della ricostruzione ed esegue il loop sugli eventi, prendendo come input il `TTree` prodotto dal codice di simulazione e creando un `TTree` in output.

- La funzione `DeltaPhi` restituisce la separazione angolare azimutali tra due punti.

- La funzione `FormTracklets` prende in input le hits sui due layer e restituisce un `vector` contenente tutte le possibili tracklets, selezionate entro il valore di Δφ di taglio definito nel file di configurazione.

RICOSTRUZIONE: MAIN (2)

- La funzione `ReconsructVertex` prende in input le tracklets generate e restituisce la coordinata Z ricostruita utilizzando il metodo della running window.

- La funzione `RunningWindow` prende in input l'ampiezza della finestra e le intersezioni di ogni tracklet con l'asse Z (ordinate per Z crescente), e restituisce la media calcolata sulle intersezioni situate nella finestra più popolata. La funzione gestisce anche un puntatore booleano che viene impostato a `true` solo se viene identificata una finestra con almeno due valori.

! Per come è implementato l'algoritmo, se la molteplicità massima è condivisa da più finestre viene considerata la finestra più a sinistra.

RICOSTRUZIONE: MAIN (3)

- Nel loop vengono eseguite le seguenti operazioni:
  - Salvataggio della molteplicità e del valore vero della coordinata Z del vertice
  - Creazione dei `vector` delle hits con applicazione dello smearing
  - Creazione delle tracklets
  - Ricostruzione del vertice con il metodo della running window. Se viene trovata una sola tracklet, come stima della coordinata Z si utilizza la singola intersezione con l'asse Z (`RunningWindow` fallisce se è disponibile una sola tracklet).

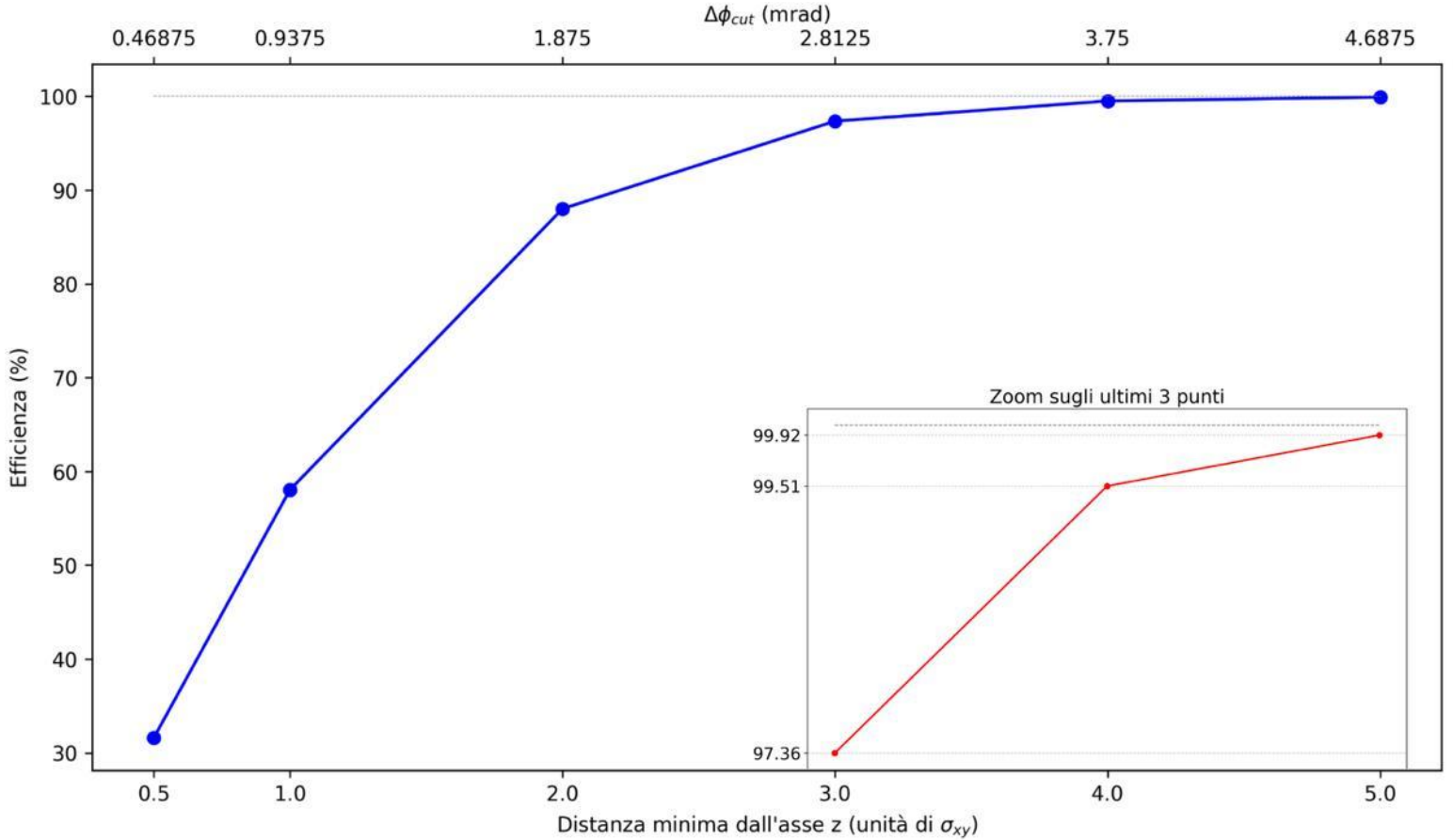
RICOSTRUZIONE: TAGLIO IN Δφ

- Il valore di taglio di Δφ scelto è 4.6875 mrad. La scelta del valore di taglio è stata effettuata calcolando la distanza di minimo approccio all'asse Z corrispondente ad un dato valore di Δφ; il valore utilizzato corrisponde ad una distanza di minimo approccio pari a 5σ, dove σ è la dispersione con cui viene generata la coordinata radiale del vertice.

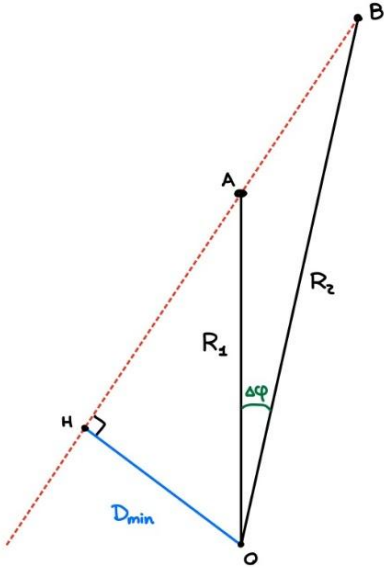
- Per testare il valore di Δφ scelto è stata effettuata una simulazione su 1'000'000 eventi con i seguenti parametri:
  - molteplicità = 1
  - vertice fisso nell'origine
  - assenza di noise
  - pseudorapidità limitata in modo da avere un'efficienza del 100%

  Il valore di Δφ utilizzato risulta avere un'efficienza del 99.92%

RICOSTRUZIONE: TAGLIO IN Δφ (2)



Relazione tra Δφ e distanza di minimo approccio



D_{min} è l'altezza del triangolo AOB relativa ad AB

$$D_{min} = \frac{2R_1R_2\sin(\Delta\varphi)}{AB} \qquad AB = \sqrt{R_1^2 + R_2^2 - 2R_1R_2\cos(\Delta\varphi)}$$

Si utilizza l'approssimazione di piccoli angoli: $\cos(\Delta\varphi) \approx 1$ $\sin(\Delta\varphi) \approx \Delta\varphi$

$$D_{min} \approx \frac{2R_1R_2\Delta\varphi}{R_2 - R_1} \longrightarrow \boxed{\Delta\varphi \approx \frac{(R_2 - R_1)}{2R_1R_2} D_{min}}$$

ANALISI
(codici)

ANALISI: MAIN

- Il main code `main_analysis.cpp` prende in input il `TTree` prodotto dal codice di ricostruzione e genera i plot come immagini `.png` nella directory `outputs/plots`, inoltre gli istogrammi vengono salvati in un file `.root` nella directory `outputs`
- Per la definizione dei valori di configurazione si utilizza la classe ROOT `Tenv`
- Nel file di configurazione è possibile decidere quali plot vengono generati impostando le relative flags su `true` o `false`.
- Le varie funzioni `Display` salvano le immagini dei plot e aggiungono gli istogrammi al file `.root`.

ANALISI: MAIN (2)

- Il primo loop sugli eventi riempie un istogramma contenente il numero di eventi simulati in funzione della molteplicità.
- Se la ricostruzione ha avuto successo, il loop riempie un istogramma contenente il numero di eventi ricostruiti in funzione della molteplicità.
- Se la ricostruzione ha avuto successo, il loop riempie gli istogrammi 2D dei residui in funzione di molteplicità e coordinata Z del vertice.
- Gli istogrammi dei residui (totali e nell'intervallo di molteplicità selezionato) vengono generati come slice degli istogrammi 2D.

ANALISI: MAIN (3)

- Gli istogrammi della risoluzione in funzione della molteplicità vengono popolati facendo slice degli istogrammi 2D ed estraendo i valori di RMS delle distribuzioni risultanti.

- L'efficienza in funzione della molteplicità viene generata con la classe ROOT `TEfficiency`, che calcola il rapporto tra i singoli bin degli istogrammi riempiti nel loop sugli eventi inserendo automaticamente errori binomiali ed evitando che le barre d'errore comprendano valori di efficienza maggiori di 1.

ANALISI: MAIN (4)

- Per creare gli istogrammi di risoluzione ed efficienza in funzione della coordinata Z del vertice, il binning viene determinato accorpando i bin dell'istogramma 2D generato nel primo loop sugli eventi in modo da avere sufficiente statistica (almeno 1000 entries per ciascuna slice).

- La funzione `FormBinEdges` prende in input l'istogramma 2D e restituisce un `vector` ordinato contenente gli estremi dei bin accorpati; l'algoritmo parte dal centro (si assume che il bin centrale abbia molteplicità sufficiente) e gestisce separatamente la parte destra (Z>0) e la parte sinistra (Z<0).

ANALISI: MAIN (5)

- Un ulteriore loop sugli eventi popola gli istogrammi contenenti il numero di eventi simulati e ricostruiti in funzione della coordinata Z del vertice secondo il binning definito in precedenza.

- L'efficienza in funzione della coordinata Z del vertice viene calcolata come rapporto tra questi due istogrammi utilizzando `TEfficiency.h`

- L'istogramma della risoluzione in funzione della coordinata Z del vertice viene popolato facendo slice dell'istogramma 2D secondo il binning definito in precedenza ed estraendo i valori di RMS delle distribuzioni risultanti.