Universidad de Nariño

Facultad de Ciencias Exactas y Naturales

Departamento de Física

Simulación de un detector de partículas segmentado para muones atmosféricos

TRABAJO DE GRADO

Para optar al título profesional de:

Físico

Jairo Hernán Rodríguez Rondón

San Juan de Pasto, Colombia

Noviembre 2016

Universidad de Nariño

Facultad de Ciencias Exactas y Naturales

Departamento de Física

Simulación de un detector de partículas segmentado para muones atmosféricos

Jairo Hernán Rodríguez Rondón

TRABAJO DE GRADO

Director:

Jaime Alfredo Betancourt

Master en Física

San Juan de Pasto, Colombia

Noviembre 2016

Nota de Aceptación

_____

_____

_____

_____

_____

_____

Jaime Alfredo Betancourt MS.c

Director

David Martínez Caicedo Ph.D

Jurado

Alex Tapia Casanova Ph.D

Jurado

San Juan de Pasto, Noviembre 2016

Simulación de un detector de partículas segmentado para muones atmosféricos

Resumen

Debido a que los muones son partículas que se producen en gran cantidad en nuestra atmósfera producto de la interacción de rayos cósmicos que la atraviesan, se propuso para este trabajo simular como los muones se distribuirían lateralmente (MLDF) y la energía con la que llegarían a la superficie de la ciudad de Pasto, para ello se implementó una simulación Monte Carlo usando el programa CORSIKA [1], el cual se especializa en simular cascadas atmosféricas extendidas (EAS) producto de la interacción de rayos cósmicos con la atmósfera. También se diseñó y simuló un detector de barras centelleadoras en el programa de interacción radiación-materia GEANT4 [2] [3], para observar como los muones interaccionan al pasar el detector, los procesos físicos involucrados, la energía que depositarían en él y como sus trayectorias se verían afectadas.

Palabras claves: *Muones, rayos cósmicos, EAS, simulación, distribución lateral, energía, CORSIKA, GEANT4, interacción radiación-materia.*

Simulation of a particle detector segmented for atmospheric muons

Abstract

Due to that muons are particles that are produced in large amount in our atmosphere by interaction of cosmic rays that across it, was proposed for this work simulate how the muons would be distributed laterally (MLDF) and the energy with which would arrive to the surface of the Pasto city, For this a Monte Carlo simulation was implemented using the CORSIKA [1] program, which specializes in simulate extended atmospheric showers (EAS) product of the interaction of cosmic rays with atmosphere. Also was designed and simulated a scintillator bar detector in the GEANT4 [2] [3] radiation-matter interaction, in order to observe how the muons interacting to pass the detector, the physical processes involved, the energy that would deposit in its and how their trajectories would be affected.

Keywords: *Muons, cosmic rays, EAS, simulation, lateral distribution, energy, CORSIKA, GEANT4, radiation-matter interaction.*

Contenido

Índice de Figuras

Capítulo 1

Rayos cósmicos

1.1 Introducción

Se les conoce como rayos cósmicos a las partículas que provienen desde el espacio exterior, el nombre de esta radiación lo acuño Robert Millikan [4], quien tras realizar sus propias mediciones concluyo que en efecto, este tipo de radiación provenía de un origen muy lejano, inclusive más allá del sistema solar. Aunque la palabra "rayos" no es la expresión más acertada para lo que en realidad son, en aquella época era muy comúnmente usada para describir tipos de radiación, un ejemplo claro de ello son los rayos α, β (1899) y γ (1903).

Su descubrimiento se remonta al año 1909, cuando Theodor Wulf, inventor del primer electrómetro, demostró con su instrumento que los niveles de radiación ionizante en la cúspide de la torre Eiffel eran mayores que los de su base [5], sin embargo, su trabajo no tuvo buena acogida en la comunidad científica de la época.

En aquel entonces se pensaba que la ionización del aire era debida a fuentes de origen natural que se encontraban en la superficie terrestre por lo cual se esperaba que mientras más se alejase de ésta, dicho nivel de radiación debería disminuir. Para comprobar ésta hipótesis, en 1912, Victor Hess, realizó una serie de experimentos en los cuales ascendió tres electrómetros de Wulf montados en un globo aerostático a 5300 metros [6], para su sorpresa encontró que dicha radiación aumentaba a medida que se ascendía. Como fuente de ésta radiación se consideró al Sol, por lo que Hess volvió a realizar estos experimentos durante un eclipse solar, sin embargo, los niveles de radiación continuaban aumentando a medida que se elevaban los instrumentos, por lo que lo descartó como única fuente de éstos; con estas observaciones concluyo: "La mejor explicación al resultado de mis observaciones viene dada por la suposición de

que una radiación de un enorme poder de penetración entra en nuestra atmósfera desde arriba".

En 1929, Kolhösther y Walter Bothe [7] realizaron un experimento con el fin de determinar si los rayos cósmicos estaban compuestos por radiación γ o por partículas cargadas. Usando dos contadores Geiger-Müller colocados uno encima del otro, notaron que la ionización simultanea de ambos equipos era muy frecuente, inclusive interponiendo materiales absorbentes entre los contadores, esto les sugirió que este tipo de radiación se trataban de partículas cargadas y altamente energéticas; según sus mediciones estimaron que sus energías estaban en el rango de 10^9 - 10^{10} eV.

En 1938, Auger en conjunto con sus colaboradores [8], a partir de la detección de un gran número de partículas que producían coincidencias en los contadores Geiger-Müller ubicados en tierra separados por una distancia de 300 metros entre si, propuso la existencia de cascada de partículas secundarias originadas al paso de los rayos cósmicos por la atmósfera, teniendo en cuenta la extensión del área en donde se realizó la detección, Auger concluyó que la energía del rayo cósmico primario debía superar los 10^{15} eV. Este hecho dio el nacimiento a la técnica de detección de superficie.

Entre 1954 y 1957, en la estación de Harvard Agassiz [9], con un arreglo de 15 contadores Geiger-Müller estimó energías de rayos cósmicos entre 3×10^{15} y 10^{18} eV, en 1961, en el experimento de Volcano Ranch [10], se obtuvieron datos de una cascada los cuales revelaban que la energía del rayo cósmico primario rondaba los 10^{20} eV. Más tarde en lo años 90's, en el observatorio Fly's Eye, el cual utilizaba una nueva técnica de detección, la fluorescencia atmosférica, registro un evento con una energía jamas observada hasta el momento, 3×10^{20} eV [11].

A pesar de contar con unos buenos avance tanto teóricos como experimentales en lo que respecta a los rayos cósmicos, aún quedan muchas preguntas por resolver, como por ejemplo ¿cuáles son sus fuentes?, ¿cómo pueden adquirir semejantes energías?, ¿cómo se propagan en el espacio antes de llegar a la Tierra?, entre otros.

1.2 Espectro de energías

El origen de los rayos cósmicos es atribuido a diversos fenómenos físicos en el universo los cuales les pueden proporcionar enormes cantidades de energía a las diversas partículas que son lanzadas al espacio, en su mayoría estas partículas suelen ser protones, no obstante entre ellas también se encuentran elementos pesados como núcleos de hierro entre otros mas pesados. Esta partículas no viajan por el espacio sin encontrar perturbaciones en su camino, una de ellas es la influencia de campos magnéticos que altera las trayectorias de dichas partículas entre otras interacciones, la combinación de estas diversas perturbaciones se ven reflejadas en la alteración del espectro de energía y sus direcciones de arribo a la Tierra.

Para el rango de bajas energías ($E \leqslant 10^9$), el espectro está dominado por los vientos solares, la magnetósfera y la heliósfera. Para energías mayores, el espectro se comporta muy bien de acuerdo a una ley de potencias:

$$\frac{dN}{dE} \propto E^{-\alpha} \tag{1.1}$$

Donde N representa el flujo de rayos cósmicos, E la energía de las partículas y α conocido como índice espectral. El índice espectral α no es fijo, éste tiene tres cambio importante, dos de ellos denominados como "rodilla" y el tercero "tobillo". La primera "rodilla" ubicada alrededor de $E \sim 10^{15.5}$eV donde α cambia de 2.7 a 3.1, la segunda "rodilla" se presenta alrededor de $E \sim 10^{17.7}$eV donde α cambia de 3.1 a 3.3. Para energías superiores a $E \sim 10^{18.5}$eV, α vuelve a tomar el valor de 2.7, siendo esta ultima zona conocida como "tobillo". Este último cambio es de mucho interés para la física ya que los procesos para que las partículas puedan adquirir semejantes energías no son muy claros. La figura 1.1 [12] muestra el espectro conocido de rayos cósmicos tomados en diferentes experimentos, además, se puede ver las energías que se pueden alcanzar el los aceleradores de partículas.

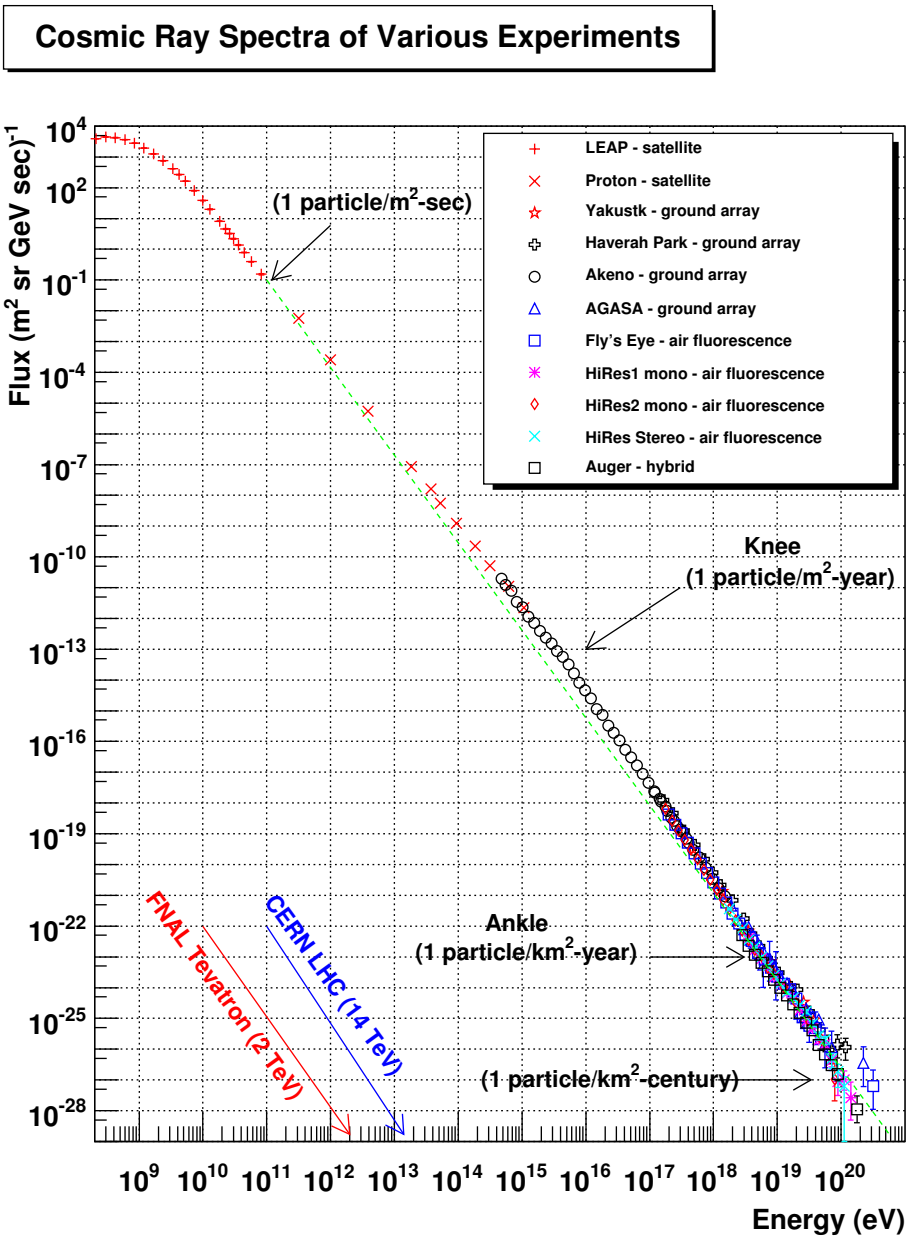

Figura 1.1: Flujo de los rayos cósmicos en función de su espectro de energía.

1.3 Composición química

Los rayos cósmicos tiene una composición muy similar a la abundancia relativa de los elementos químicos encontrados en el sistema solar, las principales diferencias se encuentran en que los rayos cósmicos son mas abundantes en Li, Be y Bo y ligeramente mas en Fe, también éstos tiene una deficiencia en H (protones) y He (figura 1.2 [13]).

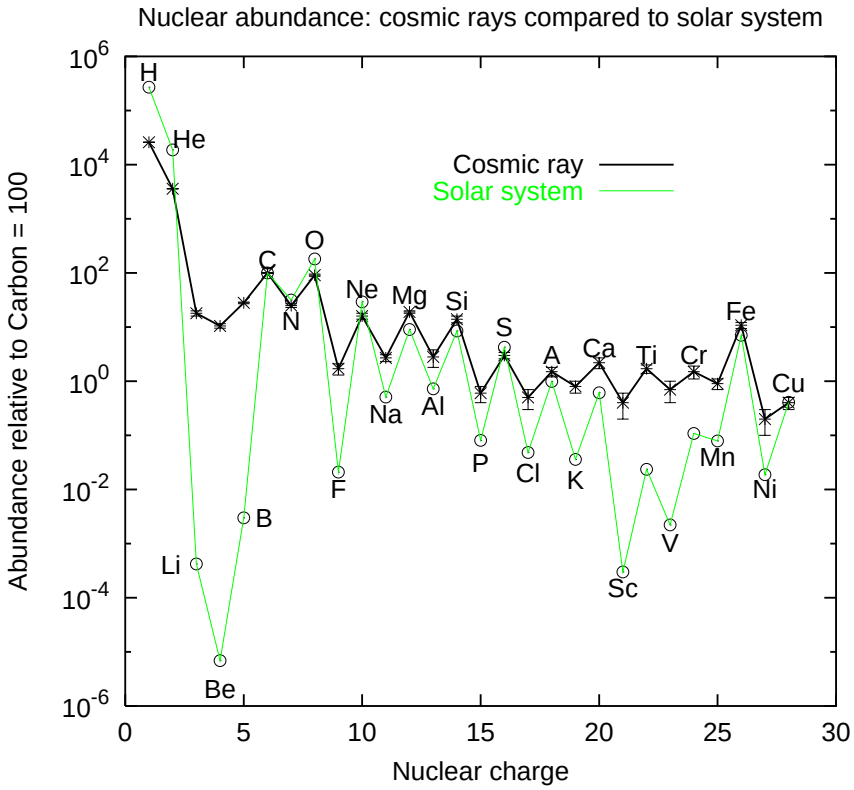

Figura 1.2: Composición de rayos cósmicos (asteriscos) y abundancia de elementos en el sistema solar (círculos).

En el rango de TeV - PeV, los rayos cósmicos se componen de 50% protones, 25% partículas α, 13% C,N,O y 12% de Fe [14], para energías superiores a éstas, debido a la baja estadística, se han recurrido a métodos de medida indirectos para poder determinar su composición, uno de ellos es por el máximo de profundidad atmosférica X_{max},aunque este método no es una prueba definitiva de la composición de los rayos cósmicos para estas energías, se ha podido considera que estén compuestos de materia ordinaria entre protones y Fe tal como se ve en la figura 1.3 [15].

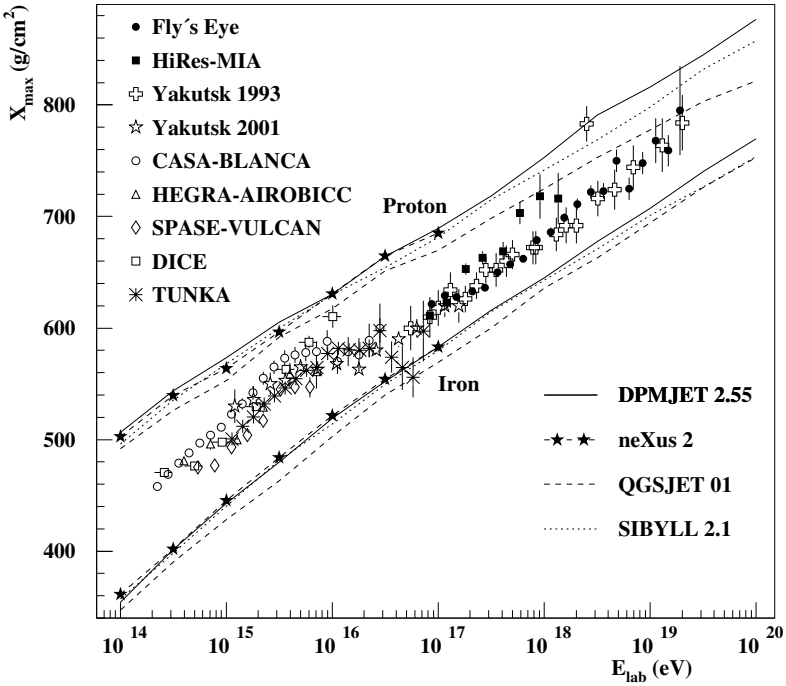

Figura 1.3: Medida de X_{max} de varios experimentos y valores esperados por simulación para primarios de protón y hierro.

Capítulo 2

Lluvias atmosféricas

2.1 Características generales

Cuando un rayo cósmico atraviesa la atmósfera, éste interactúa con moléculas presentes como N_2, O_2 y CO_2, dichas interacciones producen un gran número de nuevas partículas conocidas como partículas secundarias, estos procesos se repite en el avance de esas partículas hasta que la energía disponible no permite nuevas creaciones y se producen fenómenos de absorción y pérdida de energía, la mayoría de ellas se propagan hasta alcanzar la superficie terrestre siguiendo de cerca la trayectoria del rayo principal o también conocido como eje de la cascada, dependiendo de la energía del rayo primario, los secundarios pueden llegar a cubrir una gran extensión de la misma. El anterior fenómeno es conocido como cascadas atmosféricas extendidas o Extensive Air Shower (EAS, por sus siglas en inglés). Debido a la imposibilidad de poder realizar mediciones directas del rayo cósmico principal, ni de las interacciones que suceden, el análisis de la cascada se debe realizar mediante la detección en superficie, estos estudios permiten conocer cuales eran las características que poseía el rayo cósmico primario antes de interaccionar con la atmósfera, como cual era su composición química y energía originales, establecer una dirección de arribo, entre otras.

2.2 Desarrollo de las cascadas

Como se mencionó anteriormente, la atmósfera juega un papel importante en la forma en como se desarrolla una cascada, ya que los elementos presentes en ella son las responsables de las interacciones y la producción de secundarios a medida que los rayos cósmicos avanzan. La densidad atmosférica permite conocer la cantidad de materia con la que estos puede interaccionar, el parámetro físico con el cual se hace un

estudio principalmente de caracterización de la atmósfera se conoce como profundidad atmosférica X y esta dada por la expresión:

$$X = \int_h^\infty \rho(h')dh' \tag{2.1}$$

Donde h es la altura medida desde el nivel del mar y $\rho(h')$ es la densidad de la atmósfera a esa altura. Debido a la inhomogeneidad en la densidad atmosférica, para estudiar este parámetro se utiliza un modelo de capas atmosféricas en las cuales la densidad de cada una de estas depende de varios parámetro como altura, temperatura, y presión. La ventaja de usar este modelo atmosférico es que nos permite establecer cuales son las zonas en donde se producen los procesos de multiplicación y absorción de partículas. La profundidad atmosférica también nos permite conocer la altura a la cual se produce el máximo de producción de partículas, a esta altura se la conoce como X_{max} y como vimos en la sección 1.3, este parámetro resulta de importancia ya que permite indirectamente establecer la composición química del primario.

Las EAS iniciadas por núcleos o nucleones se dividen en tres componentes: la hadrónica, la electromagnética y la muónica, las EAS iniciadas por fotones generan principalmente componente electromagnética (figura 2.1).

2.2.1 Componente hadrónica

Cuando el rayo cósmico primario es un protón, partículas α o núcleos atómicos más pesados, la cascada que formara sera de tipo hadrónico. Las primera interacciones entre un hadrón y los núcleos atómico presentes en la atmósfera generan piones ($\pi^{\pm}, \pi^0$) los cuales constituyen las 2/3 partes de partículas producidas durante una colisión, seguidas de los kaones ($K^{\pm}, K^0$) con un 10% y el restante conformado por núcleos y nucleones, incluyendo protones y neutrones que también son producidos.

2.2.2 Componente electromagnética

Cuando una cascada es iniciada por un hadrón, como producto de las primeras interacciones los π^0 son los mayores responsables de la componente electromagnética

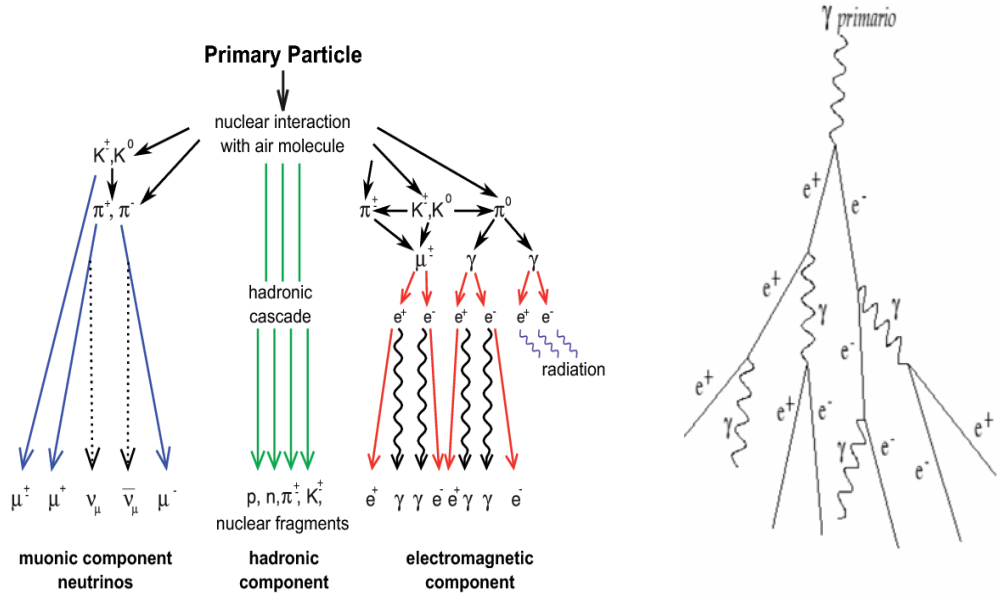

Figura 2.1: (a) Componentes de una EAS originada por núcleos o nucleones.
(b) EAS iniciada por un fotón.

en una EAS. Debido a su corta vida media ($\sim 8.4 \times 10^{-17} s$), la mayoría de estas partícula decaen antes de poder interactuar, los principales canales de decaimiento de estas partículas son:

$$\pi^0 \rightarrow \gamma + \gamma \qquad [98.8\%]$$

$$\pi^0 \rightarrow \gamma + e^+ + e^- \qquad [1.2\%]$$

Por cada una de las interacciones de los π^0, 1/3 de la energía es transferida a la parte electromagnética, a medida que las interacciones van aumentando hasta alcanzar el máximo de la cascada, aproximadamente 90% de la energía total es transferida a la componente electromagnética y termina siendo disipada en la atmósfera mediante procesos de ionización. Aproximadamente el 99.99% del total de las partículas que llegan a la superficie terrestre hace parte de la componente electromagnética de las cuales el 90% corresponde a fotones y el restante se divide entre e+ y e-.

2.2.3 Componente muónica

2.2.3.1 Muón

En 1936, el físico Carl Anderson estudiando los rayos cósmicos y catalogando las partículas detectadas, encontró una partícula que hasta la fecha no había sido observada, según las observaciones realizadas, esta partícula tenía carga negativa, pero se diferenciaba del electrón por ser mucho más masiva, pero menos que el protón. Las características de esta partícula en principio le sugirieron a Anderson de que lo que había descubierto se trataba de una de las partículas teóricas propuesta por Hideki Yukawa, los mesones, partículas que se caracterizaban por tener tener una masa media entre el electrón y el protón.

El primer nombre dado a esta partícula fue mesotrón, el cual usaba la raíz griega meso "medio" por poseer masa entre el electrón y el protón, sin embargo, más tarde se descubrieron otras partículas entre este rango de masa, a este grupo de partículas se les dio el nombre de mesones, por lo que se tuvo que modificar su nombre a mesón-μ, μ por la "m" griega, ya que esta había sido la primera partícula de masa media. Mas tarde, se observo que el mesón-μ se comportaba de manera muy diferente que los otros mesones descubiertos, al ser una partícula inestable con una corta vida media, este se desintegraba en un electrón, un neutrino y un anti-neutrino, contrario a los otros mesones que se habían observado, los cuales solo producían ya sea un neutrino o un anti-neutrino, por lo que su clasificación como mesón ya no era del todo precisa. Después de que los mesones fueran clasificados como hadrones (partículas formadas por quarks unidos por interacción nuclear fuerte), el mesón-μ perdía completamente las características de mesón, ya que se había descubierto que el no estaba conformado por quarks y tampoco experimentaba interacción fuerte, de este modo, el mesón-μ paso a llamarse muón.

2.2.3.2 Producción de muones en EAS

Existen dos formás en las que se generan muones en una EAS, ambas se deben al decaimiento de las partículas generadas en la componente hadrónica ($\pi^{\pm}$, $K^{\pm}$). Los $\pi^{\pm}$

al no interaccionar con otros núcleos atmosféricos ellos decaen de la siguiente manera:

$$\pi^{\pm} \rightarrow \mu^{\pm} + \nu_{\mu}(\overline{\nu}_{\mu}) \qquad [99.99\%]$$

Por otra parte tenemos que los $K^{\pm}$ decaen así:

$$K^{\pm} \rightarrow \mu^{\pm} + \nu_{\mu}(\overline{\nu}_{\mu}) \qquad [63.5\%]$$

Debido a la escasa interacción que los muones tiene con la materia, el número de muones generados en el máximo de la cascada se atenúa lentamente y su energía se disipa lentamente a través de procesos de ionización. A esta energía que poseen junto con la de los neutrino se le denomina energía invisible de la cascada que además constituyen el 10% restante de la energía total de la cascada. Cuando la energía que lleva un muón es demasiado baja para permitir nuevas interacciones o éste alcanzan el limite de su vida ($\sim 2.2 \times 10^{-6} s$), decaen en electrones y neutrinos que a su vez alimentan a la componente electromagnética.

2.3 Perfil lateral

El perfil lateral es la manera en como los secundarios de las EAS se distribuyen alrededor del eje de la cascada, la distribución de estas partículas se debe principalmente a dispersiones coulombianas y en menor medida a decaimientos y procesos de interacción. Las partículas de la componente hadrónica y las partículas de alta energía, permanece muy cercanas al eje de la cascada y su dispersión es del orden de metros, mientras que electrones y fotones, por ser mas susceptibles a tener este tipo de dispersión, pueden llegar a alejarse varios kilómetros del eje de la cascada. La distribución lateral de las partículas tiene simetría radial con respecto al eje de la cascada y decrece a medida que la distancia aumenta con respecto al mismo. En la distribución lateral para electrones se la obtiene aproximadamente mediante la distribución de Nishimura-Kamata-Greisen [16] [17]:

$$\rho_e(r, s, N_e) = \frac{N_e}{r_M^2} \frac{\Gamma(4.5 - s)}{2\pi\Gamma(s)\Gamma(4.5 - 2s)} \left(\frac{r}{r_M}\right)^{s-2} \left(1 + \frac{r}{r_M}\right)^{s-4.5} \tag{2.2}$$

Donde N_e es el número de electrones, s es el parámetro conocido como edad de la cascada y r_M es el radio de Molière. Este mismo análisis puede ser extendido para encontrar la distribución de otras partículas en la cascada.

El primer modelo para la distribución lateral de muones fue propuestos por el mismo Greisen el cual es:

$$\rho_\mu = N_\mu \left(\frac{r}{r_0}\right)^{-3/4} \left(1 + \frac{r}{r_0}\right)^{-5/2} \tag{2.3}$$

Donde $r_0 = 320m$ y N_μ es una constante de normalizacion. Más tarde la colaboración KASCADE propuso una nueva variante:

$$\rho_\mu = N_\mu \left(\frac{r}{r_0}\right)^{-\alpha} \left(1 + \frac{r}{r_0}\right)^{-\beta} \left(1 + \left(\frac{r}{10r_0}\right)^2\right)^{-\gamma} \tag{2.4}$$

Donde N_μ, r_0, α, β, y γ son los parámetros de ajuste.

2.4 CORSIKA: Software para simular EAS

El análisis de los datos experimentales de una EAS o la planificación de su correspondiente experimento requiere un modelo teórico detallado de la cascada que se desarrolla cuando una partícula de alta energía entra en la atmósfera. Esto sólo se puede lograr mediante cálculos detallados de Monte Carlo, teniendo en cuenta todos los conocimientos de la física de altas energías y las interacciones electromagnéticas. Para ellos se ha desarrollado una serie de programas de ordenador los cuales permiten simular el desarrollo de EAS iniciadas por rayos cósmicos de altas energías, las interacciones que estos tienen con lo elementos presentes en la atmósfera y los secundarios producto de decaimientos de partículas, CORSIKA (COsmic Ray Simulation for KASCADE) es uno de esto programa. CORSIKA permite hacer una simulación no solo con parámetros de dirección de arribo, energías y composición de rayos cósmicos, sino también nos permite implementar parámetros como el tipo de atmósfera en donde se desarrollan las EAS, campo magnético de la zona, altitud del lugar donde se desea realizar la detección de las partículas que deseemos, entre otros.

Capítulo 3

Detectores de Centelleo

Como todo detector de radiación, un centelleador es un material absorbente, el cual tiene la propiedad adicional de convertir en luz una fracción de la energías depositada por la ionizante. Las partículas cargadas interactúan con el material centellador siguiendo mecanismos bien conocidos de la interacción de la radiación en la materia. Las partículas cargadas interactúan continuamente con los electrones del medio centellador a través de interacciones coulombianas resultando en átomos excitados o ionizados.

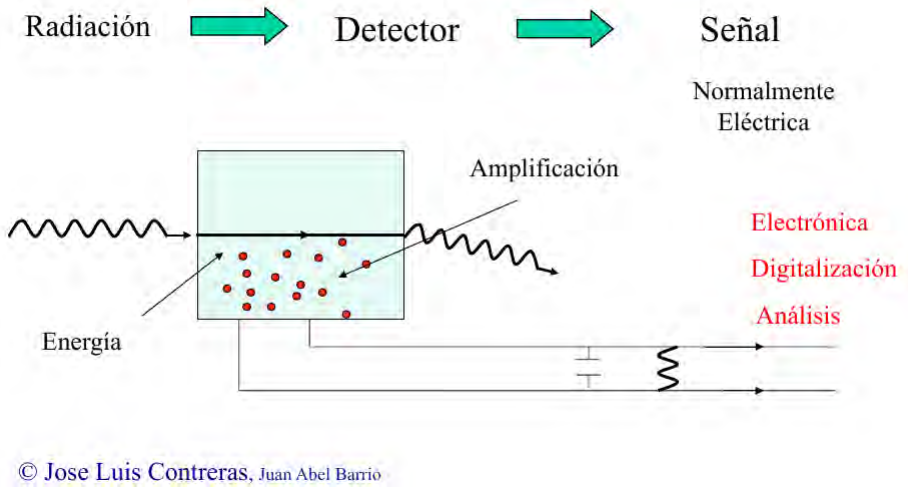

Figura 3.1: Esquema de sistema de detección.

Hay dos clases principales de centelleadores: inorgánicos y orgánicos. Para los sistemas inorgánicos (generalmente cristales iónicos), el centelleo surge de electrones y huecos, movidos de la parte inferior de la banda de conducción o la parte superior de la banda de valencia respectivamente, producidos inicialmente por la dispersión de los portadores de carga rápida.

Para los sistemas orgánicos, el centelleo surge en la transición entre un nivel

molecular excitado y el estado fundamental electrónico correspondiente. Los inorgánicos son generalmente más brillante pero con un tiempo de desintegración más lenta que orgánico. Sin embargo no existe ningún material ideal y la elección de un centelleador depende de la aplicación, ya que generalmente se acciona mediante un compromiso entre un número de parámetros físico-químicas y ópticos, tales como la densidad, las propiedades de centelleo y resistencia a la radiación. El costo de producción y procesamiento también es un problema importante teniendo en cuenta los grandes volúmenes requeridos para algunas aplicaciones.

3.1 Física de los Detectores de Centelleo

Existen dos clases de centelleadores orgánicos, los puros y los compuestos. Los centelleadores orgánicos puros son aquellos compuestos solo por una clase de molécula de la forma C_xH_y. Cuando la radiación atraviesa el material, parte de su energía puede ser absorbida excitando un electrón a un estado del singlete S superior ($S_{00} \rightarrow S_{1X}, S_{2X}, \dots$). La molécula se des-excita rápidamente al estado S_{10} sin emitir radiación (la energía se convierte en modos vibracionales) y a continuación se des-excita al estado S_{0X} emitiendo luz, a este tipo de proceso de emisión de luz se le conoce como fluorescencia y su tiempo de emisión es alrededor de $1 \sim 10$ns, su longitud de onda emitida es aproximadamente de $\lambda \sim 400$nm (Figura 3.2 [18]).

Otra manera en la que se emite luz en un centelleador es cuando la energía del estado singlete S_1 decaiga al estado triplete T_1. Estos estados posteriormente pueden des-excitarse a un estado S_{0X} en donde se emite luz como producto de su des-excitación, a este tipo proceso de emisión se le conoce como fosforescencia, este tipo de luz tiene un tiempo de emisión mayor que la producida por la fluorescencia, su tiempo de emisión esta alrededor de $1\mu s \sim 1$ms, su longitud de onda emitida es mas larga en comparación con la emitida en la fluorescencia debido a que T_1 tiene un menor nivel de energía, esta es aproximadamente de $\lambda \sim 500$nm (Figura 3.3 [18]).

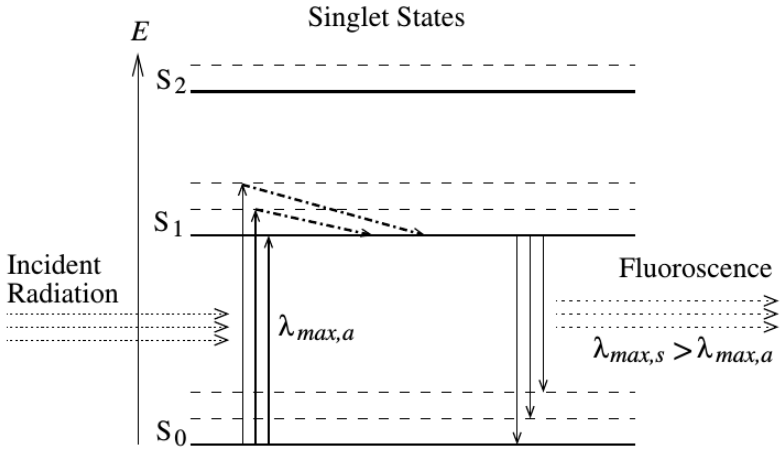

Figura 3.2: Centelleo por fluorescencia.

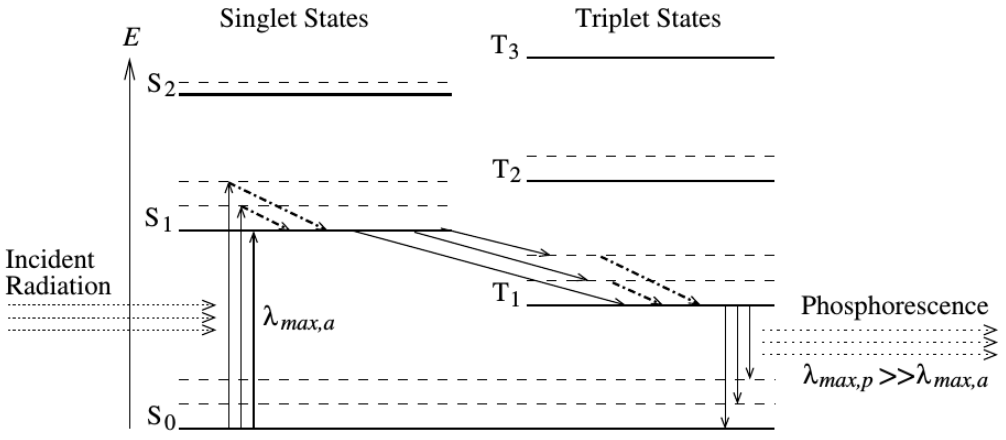

Figura 3.3: Centelleo por fosforescencia.

Un centelleador orgánico compuesto se conforma por una mezcla de varias moléculas diferentes, la energía que estas ganan producto del paso de radiación, puede ser transferida de una a otra antes de que ocurra la des-excitación. Los centelleadores compuestos son fabricados para obtener una mayor eficiencia al momento de emitir luz. Principalmente se componen de dos tipos de moléculas conocidas como solvente y soluto (también denominado "flúor"). Al paso de radiación, parte de su energía es absorbida por las moléculas del solvente X excitándola a un estado del singlete S superior ($S_{00} \rightarrow S_{1X}, S_{2X}, \dots$). La molécula se des-excita rápidamente al estado S_{10}, después de esto se pueden presentar 2 situaciones:

- ∇ Se des-excite del estado $S_{10} \rightarrow S_{0X}$ del solvente X emitiendo luz (florescencia), estos fotones son absorbidos excitando al estado singlete S del flúor Y $S_{00} \rightarrow S_{1X}$, después se des-excita el flúor Y y re-emite luz (florescencia).

- ∇ La energía absorbida por la molécula del solvente X puede ser transferida a la molécula de flúor Y sin radiar mediante el mecanismo de transferencia de energía de resonancia de Förster (FRET, por su siglas en ingles). La energía transferida excita al estado singlete S_1 del flúor Y, la cual luego se des-excita emitiendo luz (fluorescencia)

Las moléculas de solvente y flúor tienen la propiedad de que la luz emitida en ambos casos por el flúor, el disolvente sea transparente para que no sea re-absorbida por él (Figura 3.4).

En los centelleadores inorgánicos, las transiciones entre los niveles de energía está determinada por la red cristalina del material. Para la elaboración de este tipo de centelleadores se usa cristales dopados, esto se hace con el fin de aumentar la probabilidad de emisión de luz en el rango visible al paso de radiación que excita a los electrones presentes en la estructura, estas impurezas se denominan activadores los cuales modifican la estructura de bandas creando estados de energía dentro de la banda prohibida, si se usa un cristal puro, el gap energético entre niveles seria muy grande, tal que se produciría una emisión de luz de alta energía fuera del rango visible (Figura 3.5 [18]).

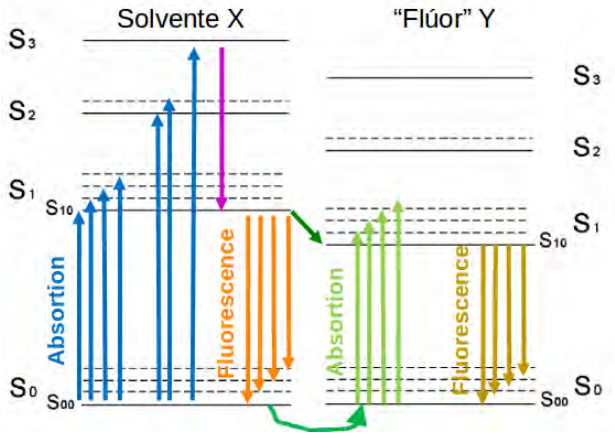

Figura 3.4: Centelleo en detectores de materiales compuestos.

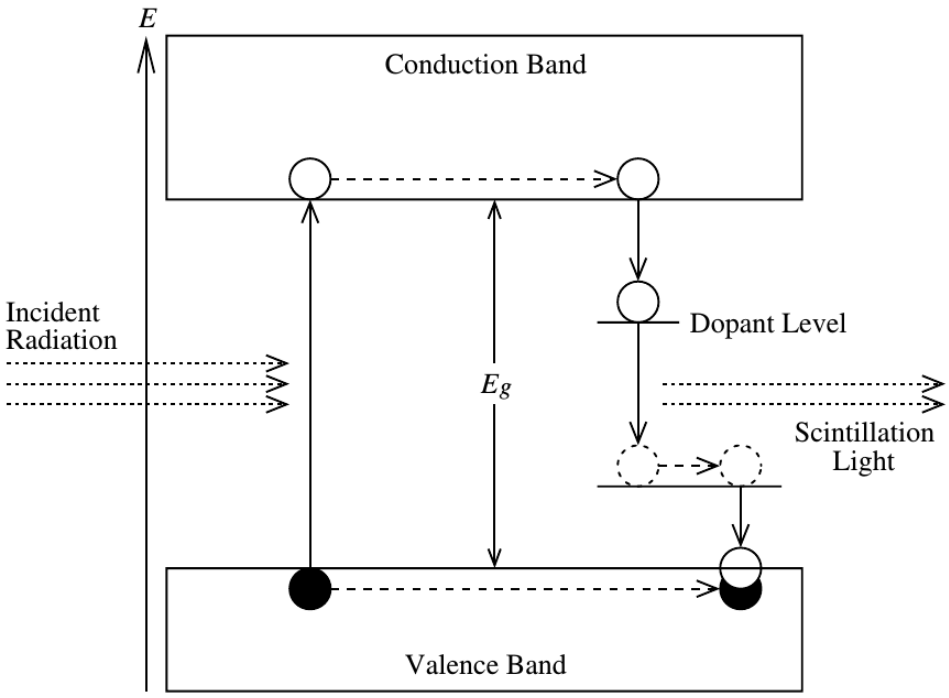

Figura 3.5: Centelleo en materiales inorgánicos

Capítulo 4

Simulación del Detector

Para estudiar el efecto que tiene el paso de muones en el detector de barras de centelleo se utilizo el paquete de simulación GEANT4 [2] [3], herramienta desarrollado en CERN que utiliza algoritmos de Monte-Carlo para reproducir los procesos que sufren las partículas al atravesar materiales.

4.1 GEANT4

El código de GEANT4 (GEometry ANd Tracking) es un paquete de software de libre distribución capaz de simular los procesos dominantes que gobiernan las interacciones de las partículas e iones con la materia, en un rango que van desde keV a TeV. GEANT4 un código Monte-Carlo escrito en C++, utiliza avanzadas técnicas de software, ingeniería y tecnología orientada a objetos capaz de simular una configuración experimental completa con todos los sensores y seguir las trayectorias de las partículas dentro de ellos. Todos los aspectos de los procesos de simulación han sido incluidos en las herramientas:

- ∇ Geometría del sistema detector.
- ∇ Materiales.
- ∇ Partículas elementales de interés.
- ∇ Generación de eventos primarios.
- ∇ Seguimiento de la trayectoria de partículas a través de los materiales y de los campos electromagnéticos.
- ∇ Procesos físicos que gobiernan las interacciones entre partículas.
- ∇ Respuesta de los componentes sensibles del detector.

- ∇ Generación de datos de los eventos.
- ∇ Almacenamiento de eventos y trazas.
- ∇ Visualización del detector y de la trayectoria de las partículas.
- ∇ Captura y análisis de datos de simulación en diferentes niveles de detalle y refinamiento.
- ∇ Interfaz con el usuario.
- ∇ Rutinas de construcción.

En el interior de GEANT4 hay una abundante cantidad de modelos físicos para manejar las interacciones de las partículas con la materia para un amplio rango de energías. Datos y experiencias han sido recogidos desde muchos lugares de todo el mundo y en este sentido, GEANT4 actúa como un repositorio que incorpora una gran cantidad de información acerca de todo lo que se conoce en cuanto a interacciones entre partículas.

4.2 Simulación en GEANT4

En cualquier simulación GEANT4 el usuario debe especificar las siguientes tres clases bases obligatorias:

- ∇ G4VUserDetectorConstruction para definir la configuración del material y la geometría del detector. Varias otras propiedades, tales como la sensibilidad del detector y los atributos de visualización, son también definidas en esta clase.
- ∇ G4VUserPhysicsList para definir todas las partículas que se van a usar, la física, procesos y parámetros de corte. (GEANT4 no define ningún proceso físico por defecto.)
- ∇ G4VUserPrimaryGeneratorAction para la generación de los vértices primarios y partículas que se van a usar.

Por otra parte hay algunas clases, por ejemplo, que permite el estudio de los puntos de interacción de partículas. Esta información por lo general se almacena en archivos con formato de ROOT [19], con el fin de posteriormente realizar el análisis de los resultados. Las clases principales que se utilizan en esta simulación son:

- UserSteppingAction para determinar la energía total depositada por procesos de pérdida de energía, los procesos y partículas involucrados en ello y las distancias recorridas por las partículas.
- G4AnalysisManager para almacenar la información en histogramas en archivos con formato ROOT.

La geometría de un detector en GEANT4 está compuesta por varios volúmenes. El volumen más grande se llama el volumen mundo. Este debe contener, con cierto margen, todos los demás volúmenes de la geometría del detector. Los otros volúmenes se crean y se colocan dentro de los volúmenes anteriores, incluidas en el volumen mundo. La forma más simple (y eficiente) para describir el mundo es una caja. Cada volumen se crea mediante la descripción de su forma y sus características físicas. Cuando un volumen está colocado dentro de otro volumen, lo llamamos volumen hija y al anterior volumen madre. El sistema de coordenadas utilizado para especificar dónde se coloca el volumen hija, es el sistema de coordenadas del volumen de la madre.

Para describir la forma de un volumen, se utiliza el concepto de un sólido. Un sólido es un objeto geométrico que tiene una forma y valores específicos para cada una de las dimensiones de esa forma. Un cubo con un lado de 10 centímetros y un cilindro de radio 30 cm y 75 cm de longitud, son ejemplos de sólidos. Para describir las propiedades completas de un volumen, se utiliza un volumen lógico. Incluye las propiedades geométricas del sólido, y añade características físicas: el material del volumen; si contiene cualquier elemento sensible del detector; el campo magnético; etcétera. Para describir cómo posicionar el volumen se crea un volumen físico.

La información física de una partícula se almacena en lo que en GEANT4 se conoce como track. Un track se lo interpreta como una "foto instantánea" de una partícula, en la cual solo almacena la información determinada en la instancia actual en que se encuentre la partícula, además estos tracks no almacenan cantidades previas.

4.3 Diseño del detector de centelleo

Para simular el detector, primero se creo un placa trapezoidal de bases 39,9 cm, longitud 40,85 cm y grosor 1,645 cm, se escogieron estas dimensiones con el fin de que al superponer las placas, las barras que se pondrán en ellas coincidan a lo largo del detector.

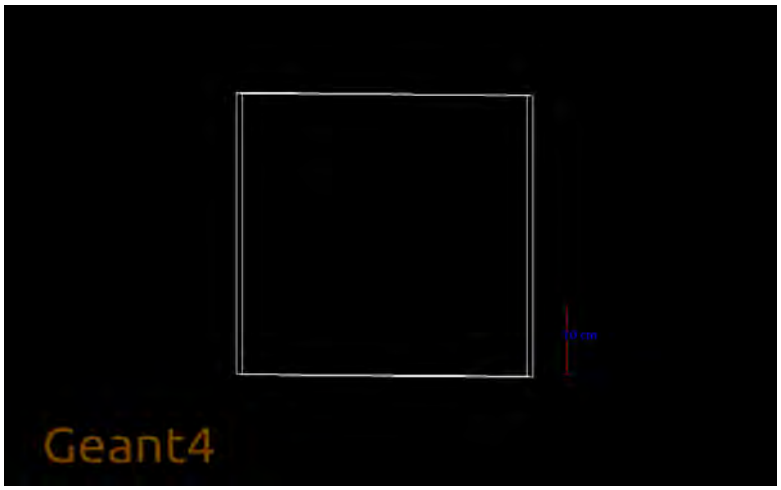

Figura 4.1: Placa trapezoidal vista frontal.

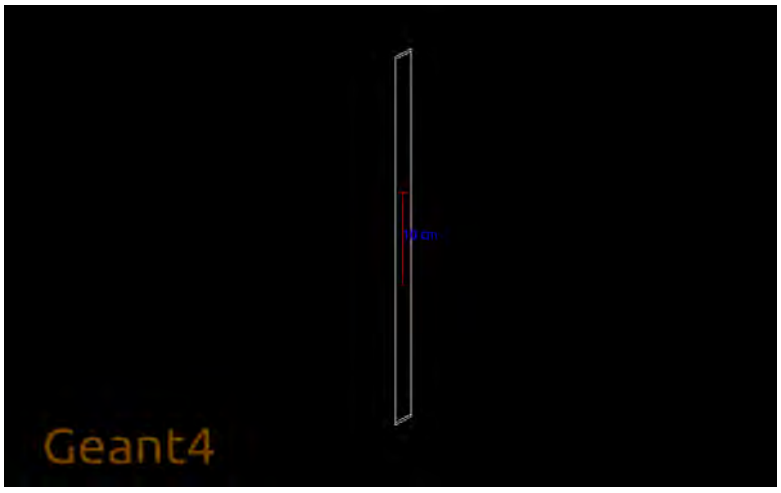

Figura 4.2: Placa trapezoidal vista lateral.

Dentro de cada plano, se creó barras trapezoidales de dimensiones de bases 1,9 cm, longitud 40,85 y grosor cm 1,645 cm, los cuales serán colocados dentro de la placa trapezoidal anterior.

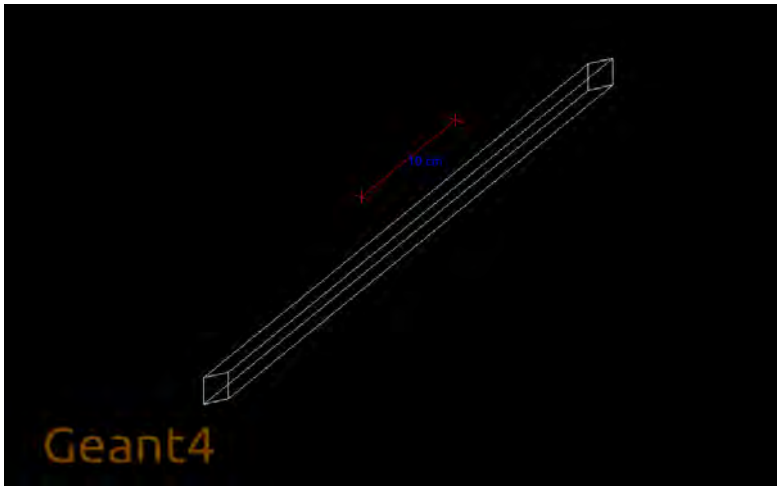

Figura 4.3: Barras trapezoidales.

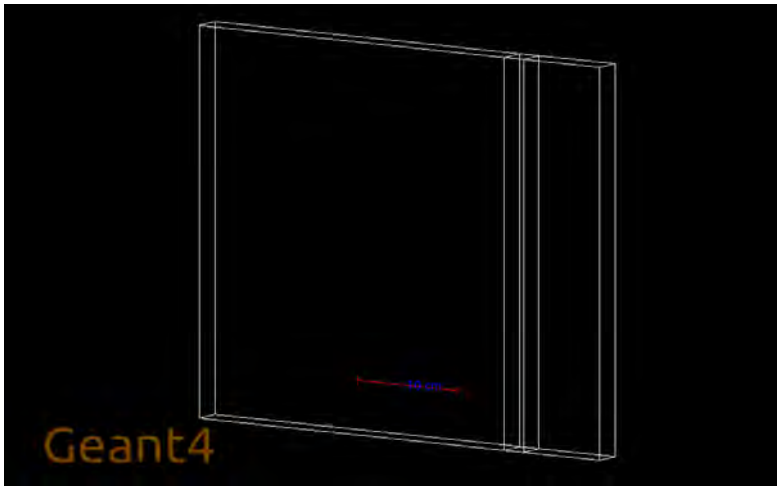

Figura 4.4: Ubicación de barras trapezoidales en los planos.

La construcción anterior debe realizarse de esa forma ya que GEANT4 posee un método de replica de volúmenes hijas, el cual nos permite realizar copias de un volumen en la dirección que se especifique, para este caso se replicó 21 veces las barras trapezoidales en la dirección X del sistema de coordenadas de GEANT4.

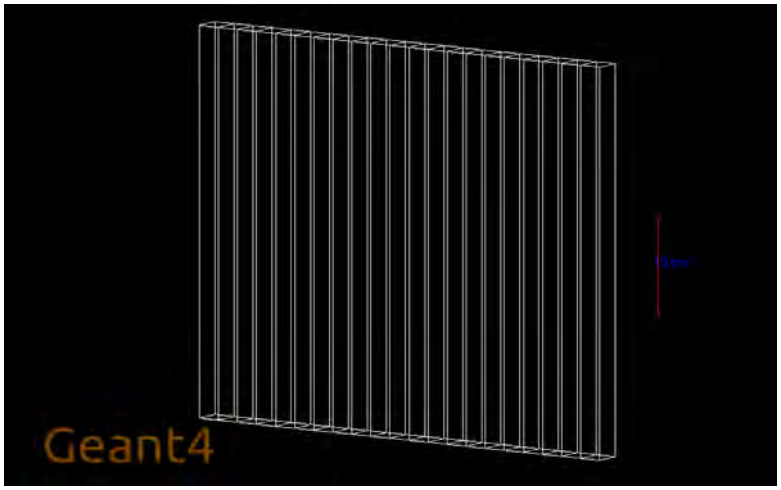

Figura 4.5: Replicas de barras trapezoidales en los planos.

Luego de crear el anterior conjunto de volúmenes, se crea las barras de centelleador plástico de poliviniltolueno (C_9H_{10}), material predefinido en GEANT4. La forma de las barras se basa en las encontradas en el experimento $Minerva$ [20] las cuales tienen base triangular. La base triangular de nuestro detector se compone de triángulos equiláteros de lado 1,9 cm y una longitud de la barra de 40,85 cm:

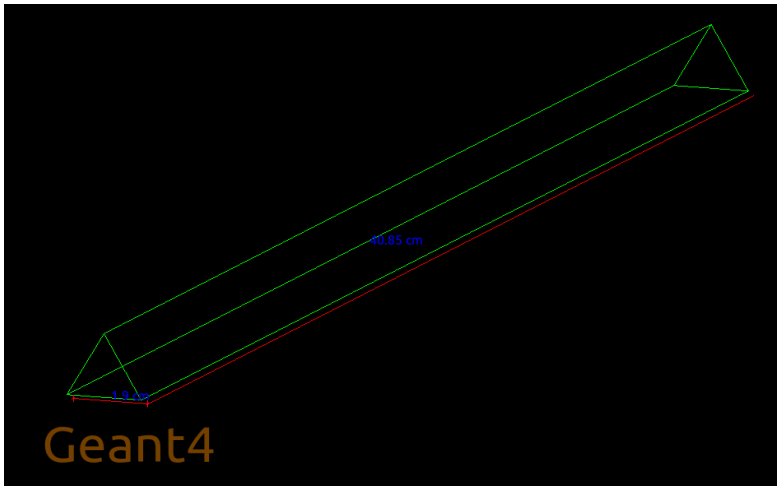

Figura 4.6: Barras centelleadoras.

Después, ubicamos cada barra de centelleador plástico dentro de las barras trapezoidales anteriormente replicada, la forma en como se ubicaran será una al lado de otra, pero una de ellas rotada 180 grados respecto la posición de la barra compañera, esto con el fin de poder ubicar dos barras centelleadoras en la barra trapezoidal.

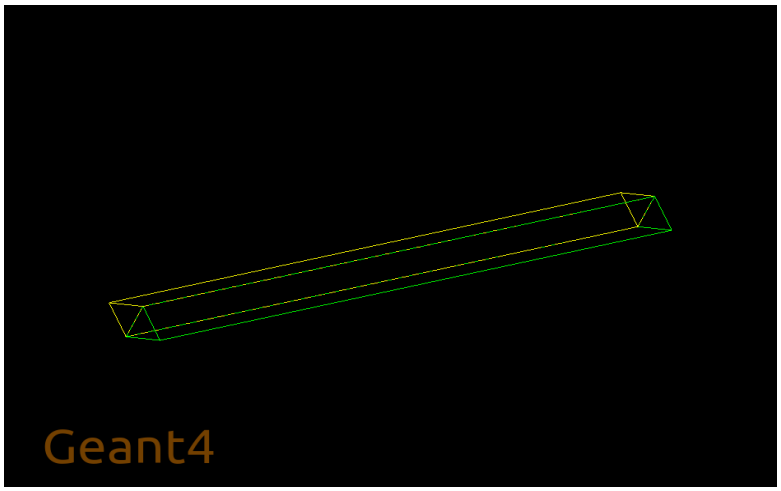

Figura 4.7: Ubicación de barras centelleadoras dentro de las barras trapezoidales.

Colocamos estos volúmenes de barra centelleadoras dentro de una barra trapezoidal y éstos se copiara a las replicas gracias al uso del método de replica de volúmenes.

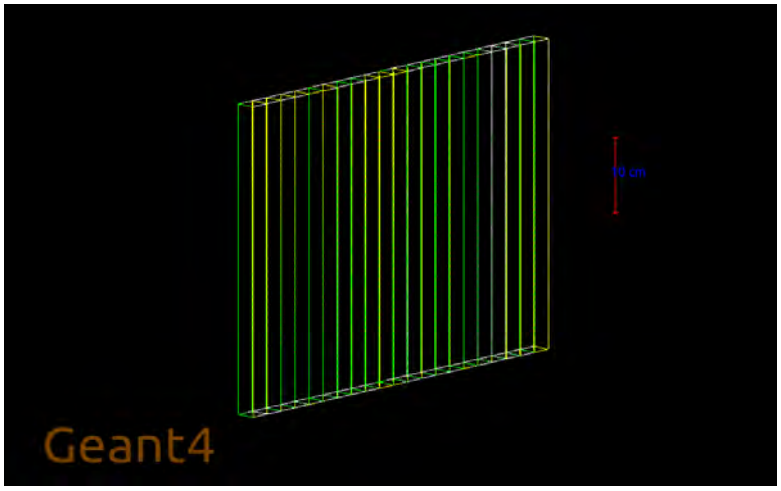

Figura 4.8: Plano de barras centelleadoras.

Ya construido el primer plano, realizamos una copia de los pasos anteriores con la diferencia de que a ésta copia se le hará una rotación de 90° respecto al eje Z.

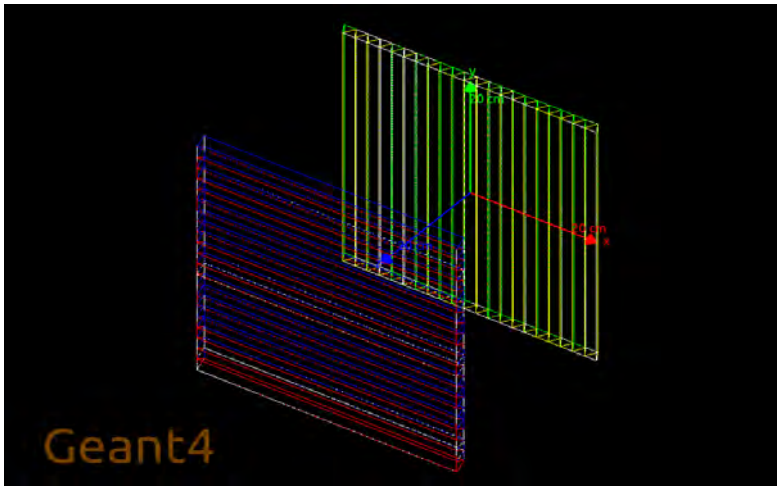

Figura 4.9: Plano de barras centelleadoras rotado.

Construidos ambos planos, finalizamos haciendo 10 copia de cada plano y los colocamos uno tras otro.

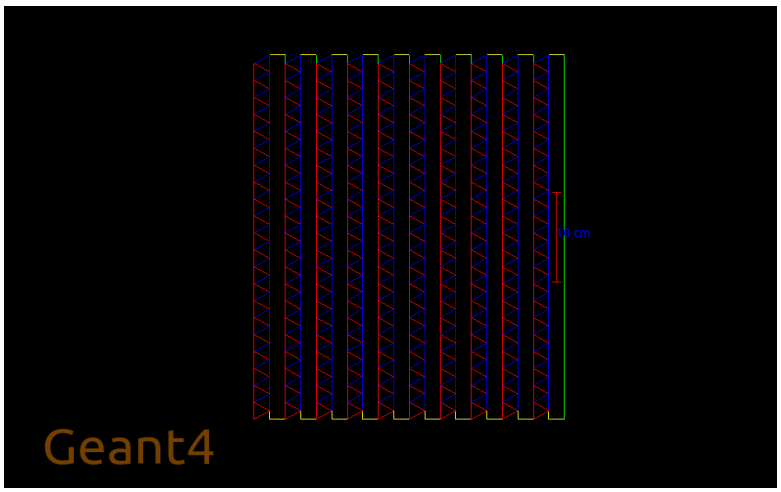

Figura 4.10: Replica de los plano de barras centelleadoras.

La forma en como se coloca los planos, nos da como resultado la segmentación o pixelado del detector. Cada pixel tiene una superficie de 9.5 mm $\times 9.5$ mm y se compone por la superposición de cuatro barras del centelleador plástico.

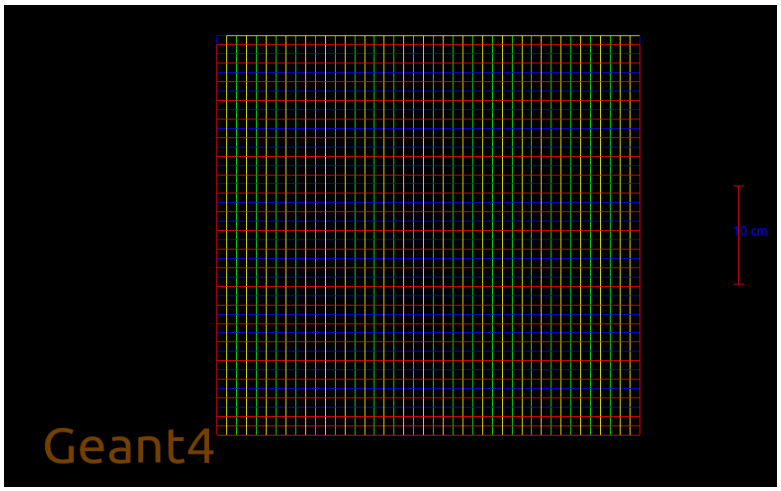

Figura 4.11: Vista total frontal del detector.

Capítulo 5

Resultados

5.1 CORSIKA

En CORSIKA [1] se realizó la simulación de EAS iniciadas por un protón con energías primarias de 5, 10 y 100 TeV y los siguientes parámetros:

- ∇ Ángulos de incidencia: 0°, 30°, 45° y 60°.
- ∇ Modelo atmosférico tropical.
- ∇ Componente de campo magnético horizontal 27,1 μT y vertical 11,8 μT.
- ∇ Altitud de la zona de observación 2530 m.s.n.m.

Se generaron generaron 200 archivos por cada ángulo para cada una de las energías del primario. Con esta información se obtuvo el número estimado de muones que arriban a 2530 m.s.n.m., (altura de la ciudad de Pasto) con los cuales se obtuvo una distribución lateral y un histograma del espectro de energías. A continuación se muestra los resultados obtenidos de la información proporcionados por la simulación.

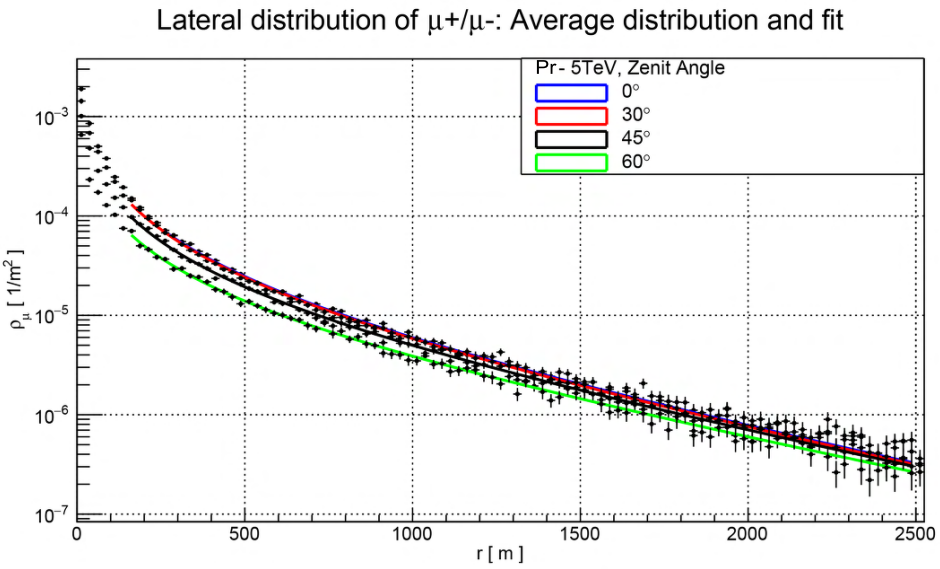

Figura 5.1: Distribución lateral de muones que arriban a la ciudad de Pasto originada por un protón con energía inicial de 5 TeV, el ajuste de los puntos se realizo según la ec. 2.4.

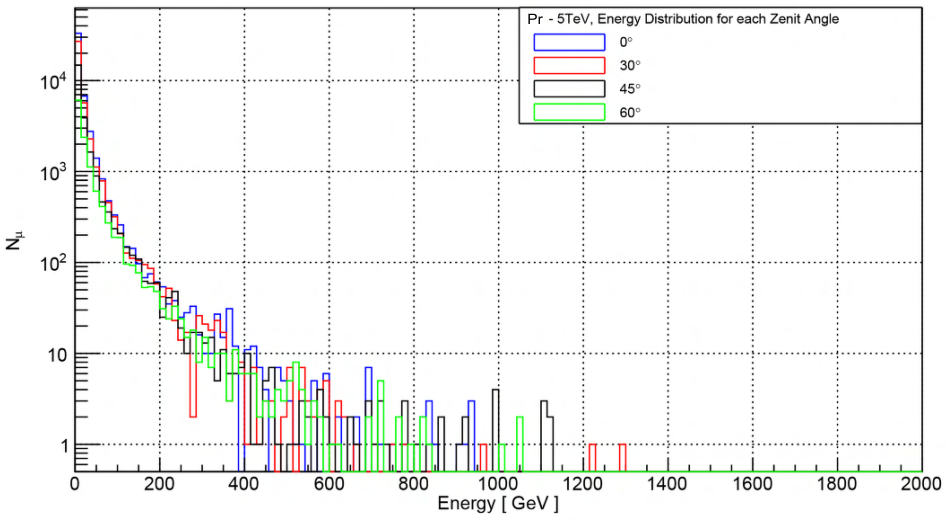

Figura 5.2: Espectro de energía de muones originados por un protón con energía inicial de 5 TeV.

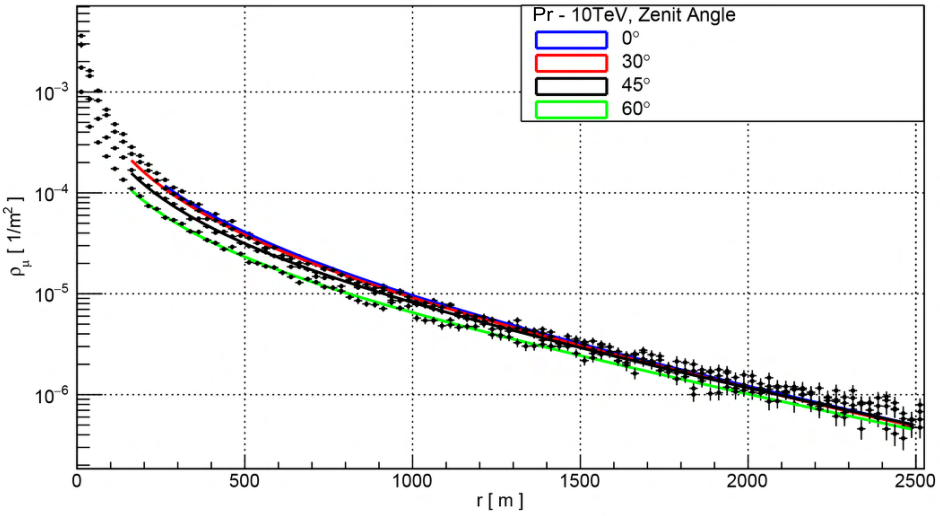

Figura 5.3: Distribución lateral de muones que arriban a la ciudad de Pasto originada por un protón con energía inicial de 10 TeV, el ajuste de los puntos se realizo según la ec. 2.4.

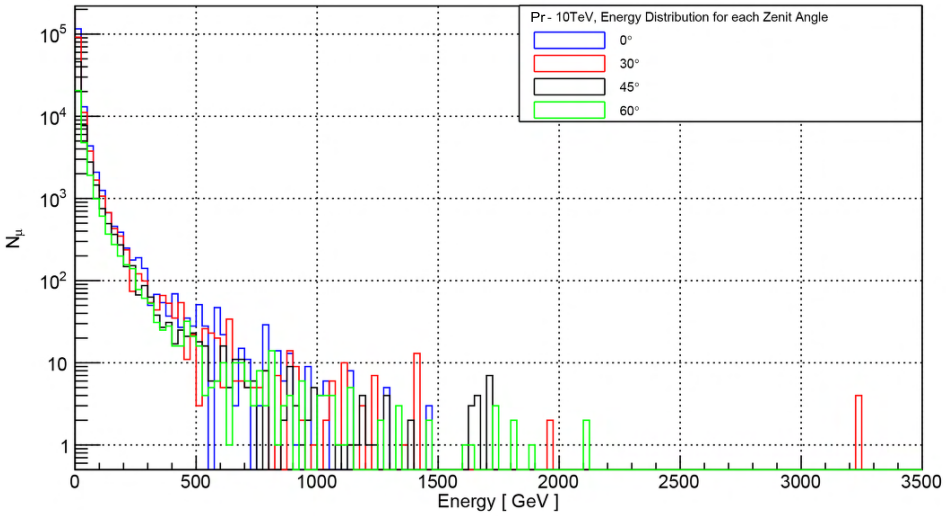

Figura 5.4: Espectro de energía de muones originados por un protón con energía inicial de 10 TeV.

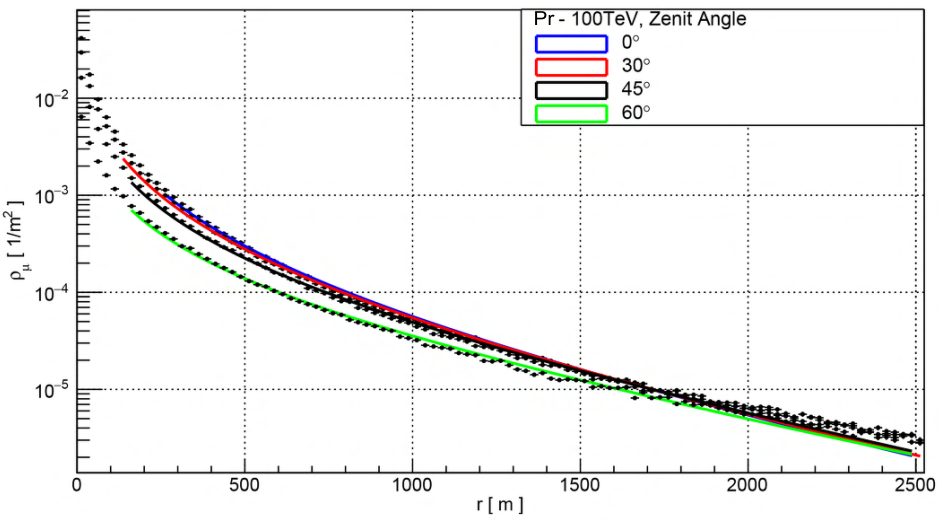

Figura 5.5: Distribución lateral de muones que arriban a la ciudad de Pasto originada por un protón con energía inicial de 100 TeV, el ajuste de los puntos se realizo según la ec. 2.4.

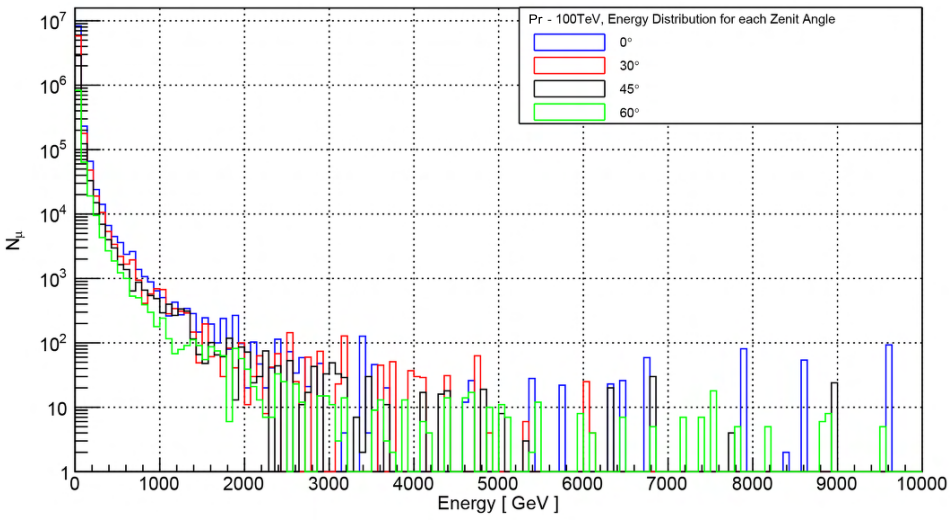

Figura 5.6: Espectro de energía de muones originados por un protón con energía inicial de 100 TeV.

La distribución lateral de muones (MLDF) fue ajustada según la ecuación 2.4, con los parámetros r_0, α, y γ fijados en 320m, 0.75 y 3 respectivamente [21]. Los parámetros N_μ y β se dejaron libres, cuyos valores después de realizado el ajuste son los siguientes:

Protón												
θ		N_μ	β	$\overline{E}_\mu(GeV)$		N_μ	β	$\overline{E}_\mu(GeV)$		N_μ	β	$\overline{E}_\mu(GeV)$
0°		0.00014	1.5	17.7729		0.00025	1.5	18.4716		0.0029	2.0	17.746
30°	5TeV	0.00014	1.4	18.4666	10TeV	0.00021	1.4	19.5733	100TeV	0.0023	1.9	18.6003
45°		0.000097	1.3	23.648		0.00015	1.3	24.9834		0.0015	1.6	24.2108
60°		0.000063	1.2	33.5182		0.000010	1.1	35.1987		0.00074	1.3	38.4453

Tabla 5.1: Parametros de ajuste para la MLDF y energias medias de muones a la altura de la ciudad de Pasto.

También se realizó el ratio entre las distribuciones laterales obtenidas así: $\rho_\mu(0°)/\rho_\mu(30°)$, $\rho_\mu(0°)/\rho_\mu(45°)$ y $\rho_\mu(0°)/\rho_\mu(60°)$, obteniendo las siguientes gráficas.

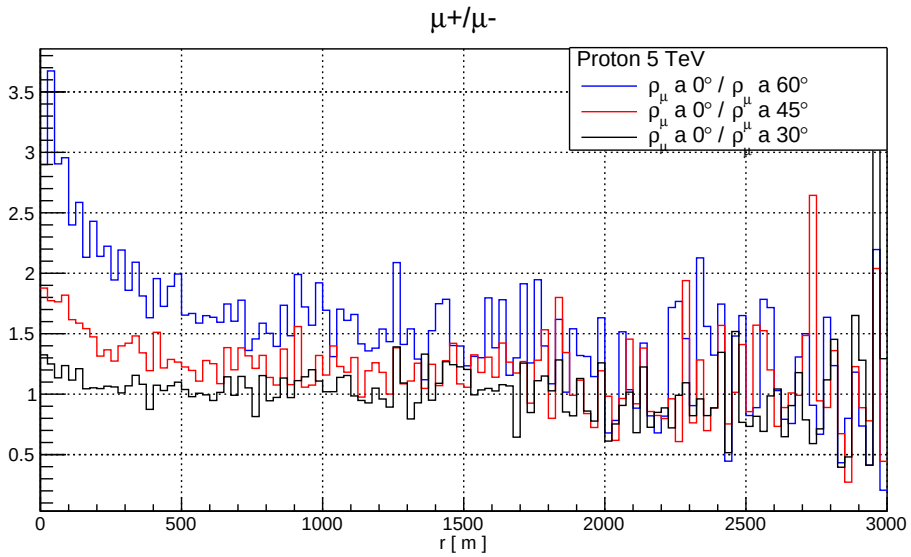

Figura 5.7: Ratio MLDF para proton primario de 5 TeV

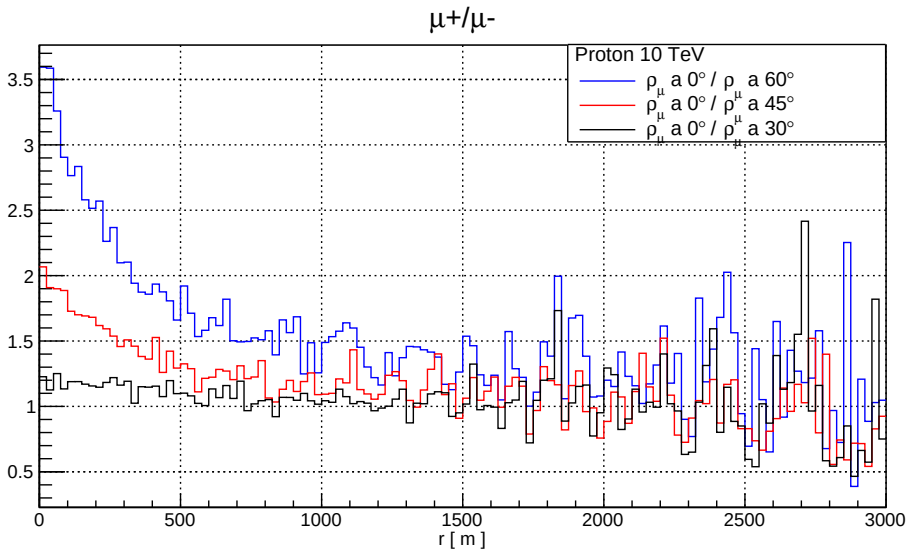

Figura 5.8: Ratio MLDF para proton primario de 10 TeV

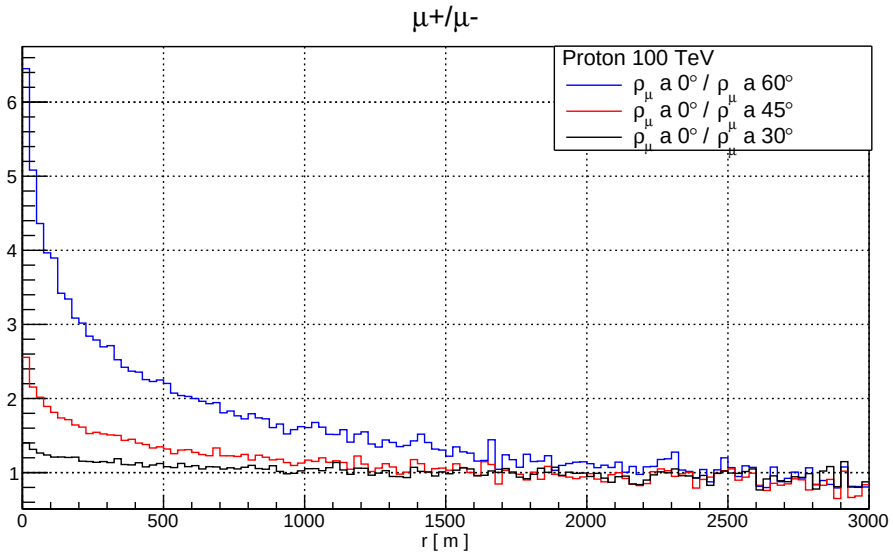

Figura 5.9: Ratio MLDF para proton primario de 100 TeV

5.2 GEANT4

Como testeo de la funcionalidad del detector se envió un flujo de 10000 muones con energía de 31 GeV, la forma del flujo para este caso es de un rectángulo de 20 cm por 20 cm, el cual abarca toda el área del detector, esta energía se la escogió en base a las energías medias de los muones obtenidos en los histogramas de energías de la simulación de CORSIKA para un protón primario de 5 TeV.

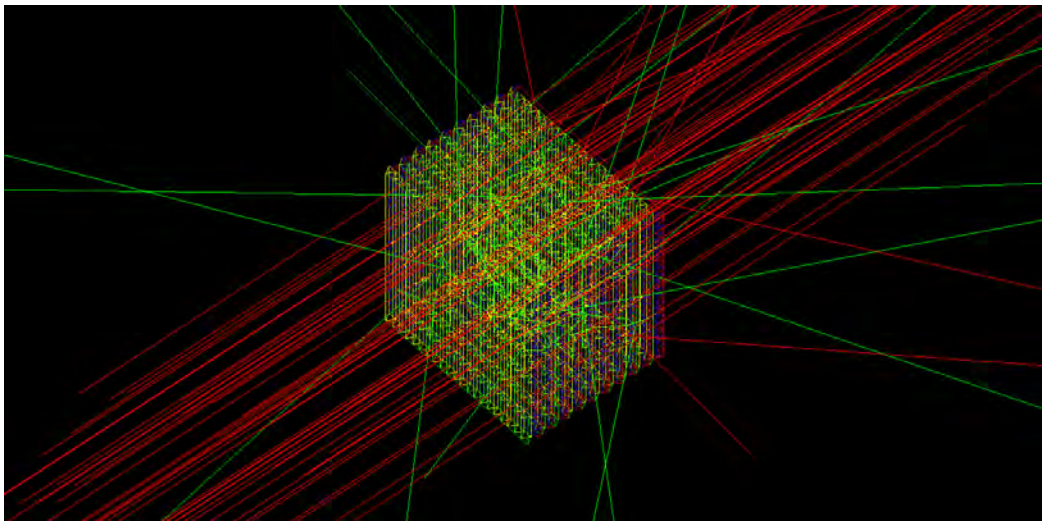

Figura 5.10: Simulación de interacciones de muones con un detector de centelleo

Se obtuvo todos los procesos correspondientes a las interacciones de los muones al paso por el detector de centelleo. Para ello se invoca los step producidos en la simulación, luego usamos el método GetPostStepPoint() para obtener el estado del actual step de la simulación, después con GetProcessDefinedStep() obtenemos los procesos sucedidos en ese step, finalmente con GetProcessName() obtenemos el nombre de los procesos ocurridos. Obtenidos los procesos sucedidos en el detector, se creó el histograma de la figura 5.11.

Para obtener el nombre de las partículas involucradas en dichos procesos, primero se invocó los steps de la simulación, luego con GetTrack() se obtuvo los tracks de las partículas en la simulación, después con GetDefinition()->GetParticleName() se obtuvo los nombres de las partículas, con ello se creo el histograma de la figura 5.12.

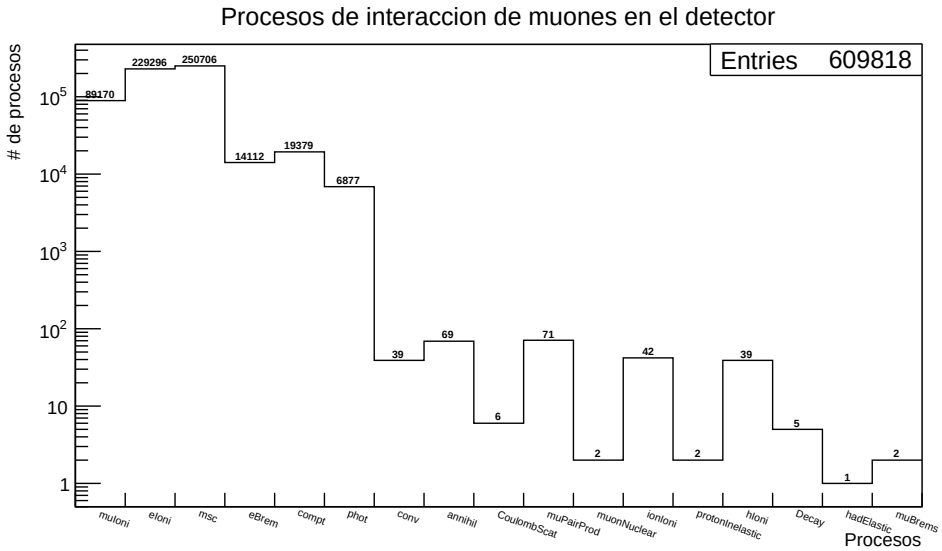

Figura 5.11: Procesos físicos producido por las interacciones de los muones con el detector centelleo.

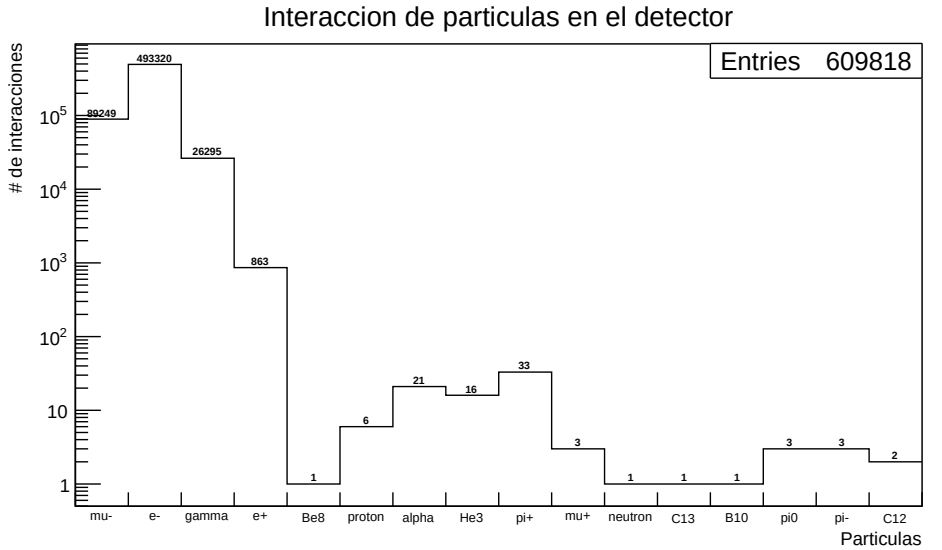

Figura 5.12: Partículas involucradas en los procesos físico que se producen en el detector de centelleo

Con la información obtenida anteriormente se realizó un histograma en dos dimensiones de la figura 5.13 de procesos físicos involucrados en función de las partículas, esto con el fin de ver más claramente que partículas son las responsables de los diferentes procesos dentro del detector.

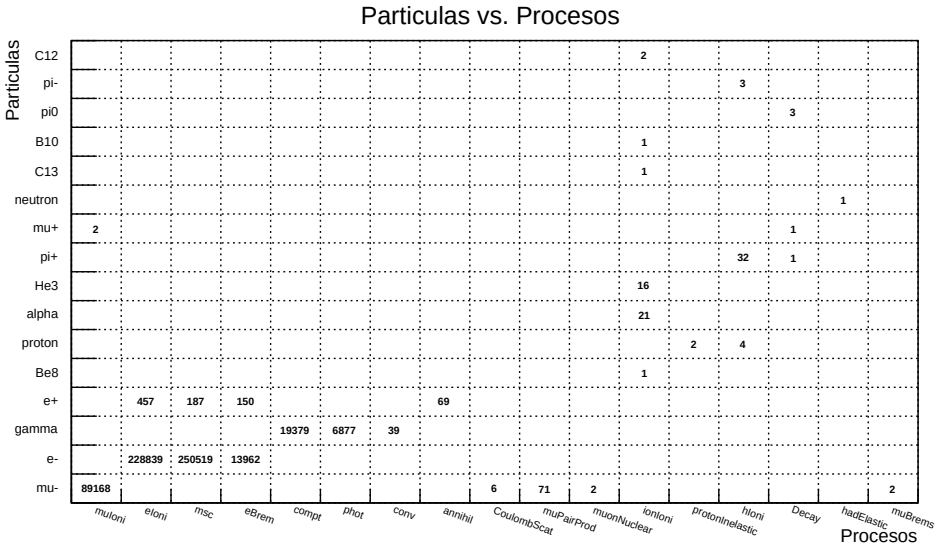

Figura 5.13: Histograma en dos dimensiones de las partículas como función de los procesos físicos generados.

También de la simulación se extrajo información sobre le energía depositada y el camino recorrido por los muones en el detector, estos resultados se pueden ver en la figura 5.14 y 5.15, respectivamente.

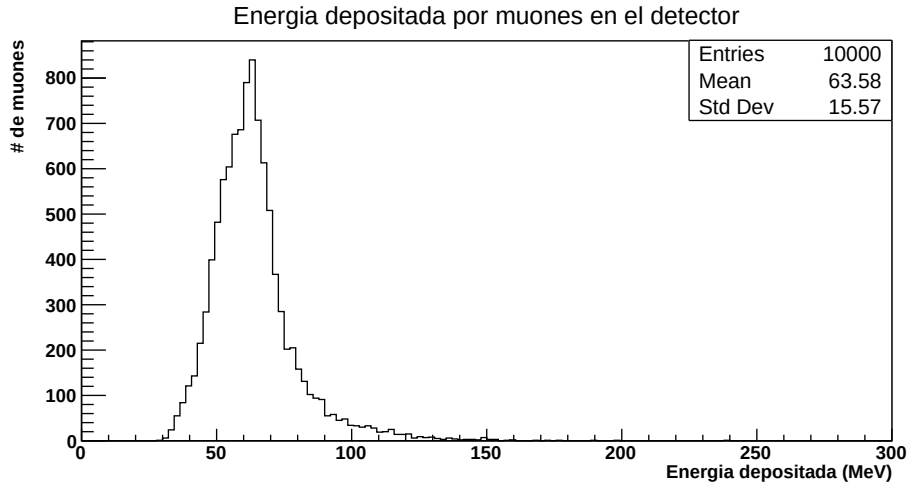

Figura 5.14: Energía depositada por muones en el detector de centelleo.

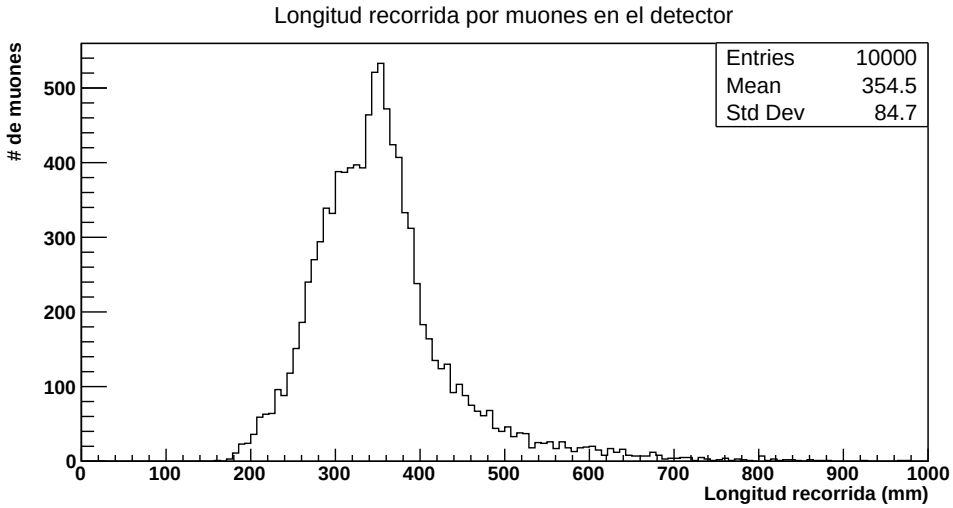

Figura 5.15: Longitud recorrida por muones en el detector.

Mediante la información de la posición de las partícula en el detector, se realizó la figura 5.16 que muestra como los muones atraviesan el detector.

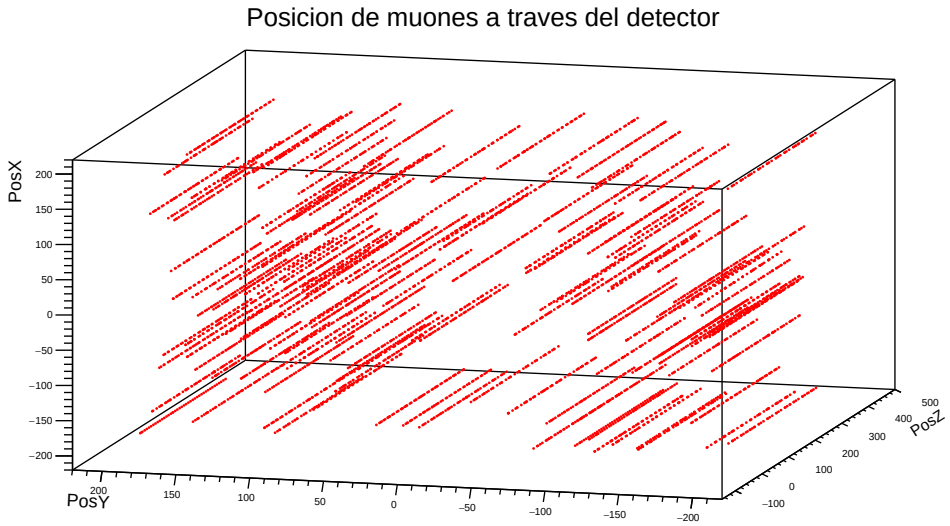

Figura 5.16: Traza de los muones en el detector.

En la figura 5.17, se muestra como los electrones producto de interacciones de los muones con el detector se mueven a través de él.

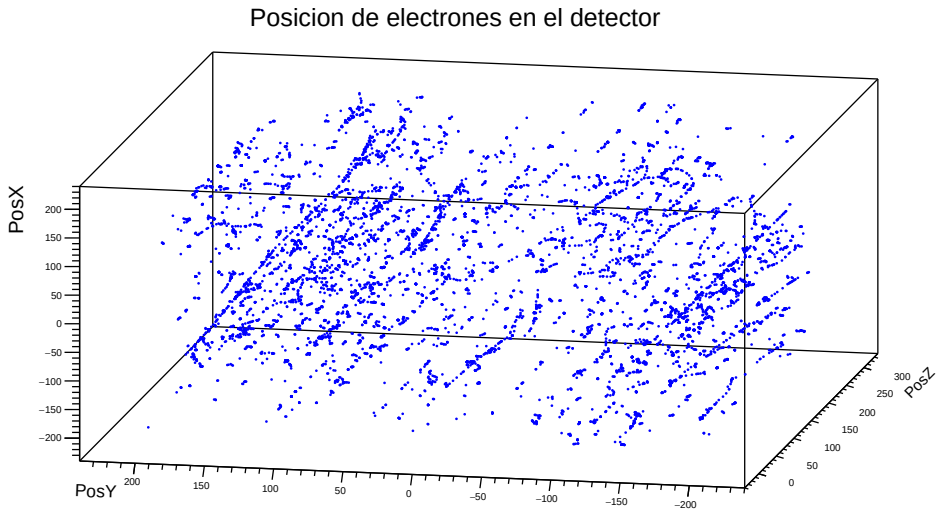

Figura 5.17: Traza de electrones producidos en el detector.

A continuación se realizó una superposición de las figuras 5.16 y 5.17 para mostrar la relación de como los muones son los causantes de que en el detector se produzcan electrones mediante los procesos descritos en la gráfica 5.13. En este caso, los electrones son producto de las ionizaciones producidas por los muones.

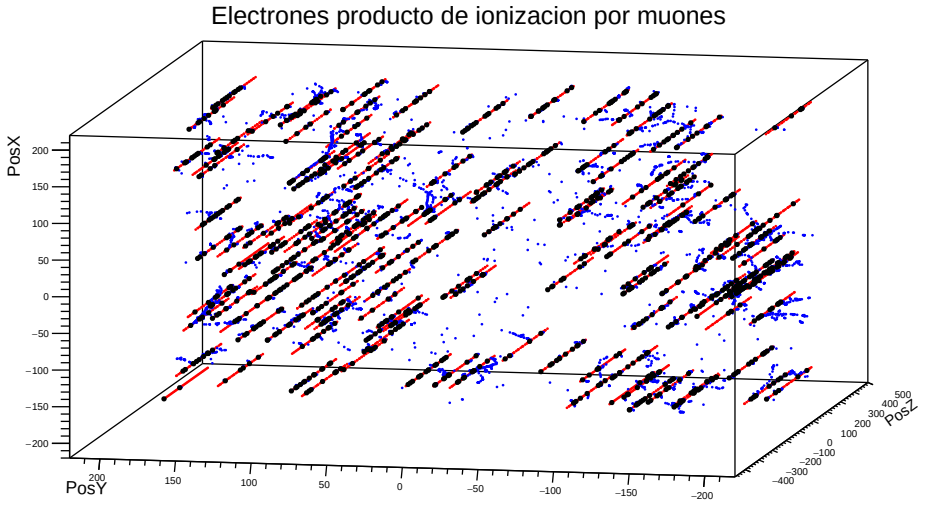

Figura 5.18: Relación entre producción de electrones (puntos azules) por proceso de ionización de muones (puntos negros) en el detector.

Por último tenemos la producción de radiación gamma (fotones de alta energía) producto del centelleo del material del detector.

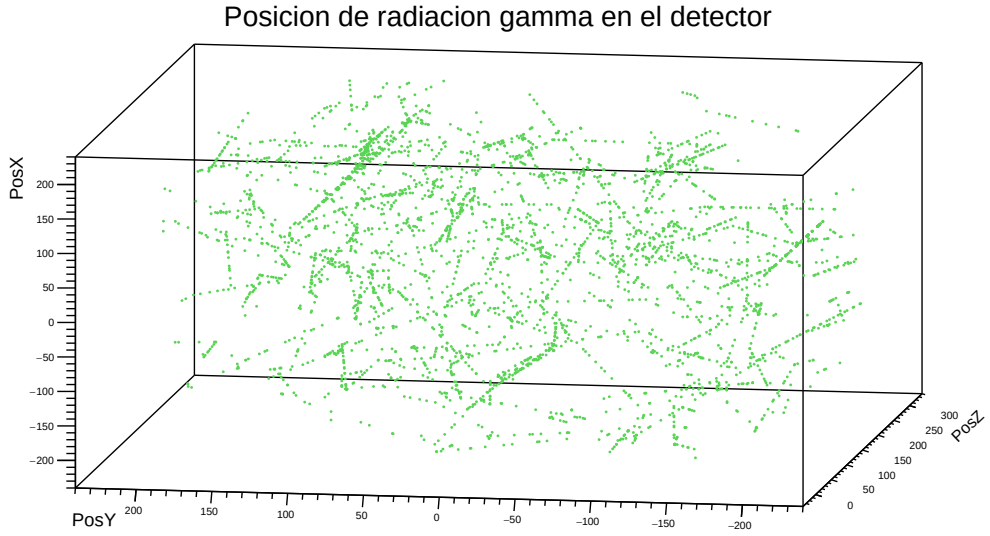

Figura 5.19: Traza de gammas en el detector.

5.3 Análisis de Resultados

- ∇ De la tabla de parámetro de ajuste para la MLDF obtenidas de las simulaciones de EAS en CORSIKA (Tabla 5.1), podemos observar que el parámetro N_μ, decrece a medida que aumenta el ángulo de incidencia del rayo cósmico primario, lo que nos sigiere que el número de muones disminuye en las proximidades del eje de la lluvia (Figuras 5.7, 5.8 y 5.9).

- ∇ De las figuras 5.11, 5.12 y 5.13, podemos ver que el mecanismo por el cual los muones pierden energía al atravesar el detector es mediante ionizaciones (muIoni) de los átomos de material del que esta compuesto, y que la energía media depositada por estas interacciones es de 63,58 $\pm$ 15,57 MeV.

- ∇ De la figura 5.15 se observa que la longitud media recorrida por los muones en el detector es de 354,5 $\pm$ 84,7 mm, lo que nos indica que los muones no interaccionan demasiado con los materiales del detector y sus trayectorias iniciales no se ven notablemente afectadas ya que la longitud total lateral del detector es de 329 mm. Esto lo podemos verificar con la figura 5.16 que nos indica las trazas de los muones en el detector, como se puede apreciar, estas son prácticamente rectas, tal y como son enviadas de la fuente.

Capítulo 6

Conclusiones y Recomendaciones

Mediante la simulación de la EAS en CORSIKA, podemos observar de una forma muy próxima a la realidad, como se produce el desarrollo lateral de los muones producidos en la atmósfera originados por rayos cósmicos, también nos permite estimar cual sera la energía de los muones que logran alcanza la superficie de observación, en este caso la ciudad de Pasto. Además mediante la simulación del detector de partículas en GEANT4, podemos estimar que tipos de procesos físicos son los que se producen en la interacción de los muones con el detector, la energía que estas partículas depositan en éste producto de los procesos físicos y como las trayectorias de éstas se ven afectadas al atravesar el detector.

En este trabajo se quiere destacar la importancia de las herramientas computacionales a la hora de evaluar la factibilidad de realizar proyectos de pequeña o gran escala sin recurrir a equipo de difícil acceso y costes económicos, teniendo resultados validos en los cuales basarnos para escoger las mejores opciones a la hora de la implementación de uno real, cuales son las ventajas y desventajas de usar ciertos materiales de detección, cuales son los mas conveniente según sea la aplicación del mismo, que tan eficientes serian a la hora de detectar las partículas que deseamos, entre otros.

Se recomienda realizar la caracterización de la atmósfera de la ciudad de Pasto para que las simulaciones de EAS implementadas en CORSIKA sean mucho mas cercanas a la realidad, ya que si bien el modelo de atmósfera tropical implementado en este caso es un buena aproximación dada la zona en la que la que la ciudad se ubica, medir adecuadamente las variables atmosféricas locales nos daría una mejor idea de como las EAS se desarrollan localmente.

Se recomienda implementar componentes ópticos al presente modelo de detector de centelleo, implementar en esta simulación una envoltura a cada una de las barras de

centelleo con un material reflectante para que los fotones producidos por el centelleo del material no escapen de una a otra barra y afecten las medidas deseadas, además de la implementación de fotomultiplicadores para que capten las señales de los fotones producidos en cada barra de centelleo y ubicarlos al extremo de cada barra para que la reconstrucción tanto de posición como energía de los muones sea mas acorde con la realidad, ya que en este prototipo la reconstrucción de energía y posición que se hizo, es debida a las posiciones "reales" y energía depositada por todos los procesos producto de la información almacenada en los datos de la simulación.

Para el flujo de muones incidentes, se recomienda implementarlo según la información extraída de las simulaciones de CORSIKA lo cual permitirá realizar medidas mas reales en en cuanto a deposición de energía y trayectoria de los muones.

La parte presentada sobre la distribución lateral de muones fue presentada como parte del póster titulado: First Monte Carlo simulation study of Galeras Volcano structure using Muon Tomography [22], en el evento ICHEP 2016 realizado en la ciudad de Chicago, EE.UU, el cual esta en procesos de publicación de proceeding en proceeding of science.

Apéndice A

Código del programa

A continuación se muestra el código de las rutinas implementadas en CORSIKA y GEANT4:

A.1 Macro CORSIKA

```
RUNNR   200                                                 run number
EVTNR   200                                                 number of first shower event
SEED    2576  0   0                                         seed for hadronic part
SEED    28837  0   0                                        seed for EGS4 part
THETAP  60    60                                            range of zenith angle (degree)
DIRECT  ~/corsika/CORSIKAWORKSPACE/cskrun/Energy_5E3/Angle_60/  output directory
ERANGE  5E3  5E3                            energy range of primary particle (GeV)
PRMPAR  14                                  particle type of prim. particle
ESLOPE  0.                                  slope of primary energy spectrum
NSHOW   1                                   number of showers to generate
PHIP    -180.  180.                         range of azimuth angle (degree)
OBSLEV  2530.E2                             Pasto observation level (in cm)
FIXCHI  0.                                  starting altitude (g/cm**2)
MAGNET  27.1  -11.8                         Pasto magnetic field (muT)
HADFLG  0  0  0  0  0  2                    flags hadr.interact.&fragmentation
ECUTS   0.05  0.05  0.000264  0.000264      energy cuts for particles (hadr. muon elec. photon)
MUADDI  T                                   additional info for muons
MUMULT  T                                   muon multiple scattering angle
ELMFLG  T   T                               em. interaction flags (NKG,EGS)
STEPFC  1.0                                 mult. scattering step length fact.
RADNKG  200.E2                              outer radius for NKG lat.dens.distr.
LONGI   T  5.  T  T                         longit.distr. & step size & fit & out
ECTMAP  25000.                              cut on gamma factor for printout
MAXPRT  1                                   max. number of printed events
THIN    1.E-06  10000.  50.                 thinning level, maximum weight factor and radial distance
THINH   1.  100.                            hadronic thinning limit
PAROUT  T  T                                output tables
USER    jairo                               user
DEBUG   F  6  F  1000000                    debug flag and log.unit for out
ATMOSPHERE 1  F                             atmospheric model
EXIT                                        terminates input
```

A.2 GEANT4

A.2.1 Headers

V0ActionInitialization.hh

```
#ifndef V0ActionInitialization_h
#define V0ActionInitialization_h 1

#include "G4VUserActionInitialization.hh"

class V0DetectorConstruction;

class V0ActionInitialization : public G4VUserActionInitialization
{
  public:
    V0ActionInitialization(V0DetectorConstruction*);
    virtual ~V0ActionInitialization();

    virtual void BuildForMaster() const;
    virtual void Build() const;

  private:
    V0DetectorConstruction* fDetConstruction;
};

#endif
```

V0Analysis.hh

```
#ifndef V0Analysis_h
#define V0Analysis_h 1

#include "g4root.hh"

#endif
```

V0DetectorConstruction.hh

```
#ifndef EDDetectorConstruction_h
#define EDDetectorConstruction_h 1

 #include "G4VUserDetectorConstruction.hh"

 class G4VPhysicalVolume;
 class G4GlobalMagFieldMessenger;

 class V0DetectorConstruction : public G4VUserDetectorConstruction
 {
   public:
     V0DetectorConstruction();
     virtual ~V0DetectorConstruction();
```

```
   public:
     virtual G4VPhysicalVolume* Construct();

    const G4VPhysicalVolume* GetAbsPsX() const;
    const G4VPhysicalVolume* GetAbsPsX1() const;
    const G4VPhysicalVolume* GetAbsPsY() const;
    const G4VPhysicalVolume* GetAbsPsY1() const;


    G4VPhysicalVolume* fAbsPsX;
    G4VPhysicalVolume* fAbsPsX1;
    G4VPhysicalVolume* fAbsPsY;
    G4VPhysicalVolume* fAbsPsY1;


};

inline const G4VPhysicalVolume* V0DetectorConstruction::GetAbsPsX() const {
  return fAbsPsX;
}

inline const G4VPhysicalVolume* V0DetectorConstruction::GetAbsPsX1() const {
  return fAbsPsX1;
}

inline const G4VPhysicalVolume* V0DetectorConstruction::GetAbsPsY() const {
  return fAbsPsY;
}

inline const G4VPhysicalVolume* V0DetectorConstruction::GetAbsPsY1() const {
  return fAbsPsY1;
}


#endif
```

V0EventAction.hh

```
#ifndef V0EventAction_h
#define V0EventAction_h 1

#include "G4UserEventAction.hh"
#include "globals.hh"

class V0EventAction : public G4UserEventAction
{
  public:
    V0EventAction();
    virtual ~V0EventAction();

    virtual void  BeginOfEventAction(const G4Event* event);
    virtual void    EndOfEventAction(const G4Event* event);

    void AddAbsPsX(G4double de, G4double dl);
    void AddAbsPsX1(G4double de, G4double dl);
    void AddAbsPsY(G4double de, G4double dl);
    void AddAbsPsY1(G4double de, G4double dl);
    void AddPosX(G4double de);
    void AddPosY(G4double de);
```

```
  private:
    G4double  fEnergyAbsPsX;
    G4double  fEnergyAbsPsX1;
    G4double  fEnergyAbsPsY;
    G4double  fEnergyAbsPsY1;
    G4double  fTrackLAbsPsX;
    G4double  fTrackLAbsPsX1;
    G4double  fTrackLAbsPsY;
    G4double  fTrackLAbsPsY1;
    G4double  fPosX;
    G4double  fPosY;

};

// inline functions

inline void V0EventAction::AddAbsPsX(G4double de, G4double dl) {
  fEnergyAbsPsX += de;
  fTrackLAbsPsX += dl;
}

inline void V0EventAction::AddAbsPsX1(G4double de, G4double dl) {
  fEnergyAbsPsX1 += de;
  fTrackLAbsPsX1 += dl;
}

inline void V0EventAction::AddAbsPsY(G4double de, G4double dl) {
  fEnergyAbsPsY += de;
  fTrackLAbsPsY += dl;
}

inline void V0EventAction::AddAbsPsY1(G4double de, G4double dl) {
  fEnergyAbsPsY1 += de;
  fTrackLAbsPsY1 += dl;
}

inline void V0EventAction::AddPosX(G4double de) {
  fPosX = de;
}

inline void V0EventAction::AddPosY(G4double de) {
  fPosY = de;
}

#endif
```

V0PrimaryGeneratorAction.hh

```
#ifndef V0PrimaryGeneratorAction_h
#define V0PrimaryGeneratorAction_h 1

#include "G4VUserPrimaryGeneratorAction.hh"
#include "globals.hh"

class G4ParticleGun;
class G4Event;

class V0PrimaryGeneratorAction : public G4VUserPrimaryGeneratorAction
```

```
{
  public:
    V0PrimaryGeneratorAction();
    virtual ~V0PrimaryGeneratorAction();

    virtual void GeneratePrimaries(G4Event*);

  private:
    G4ParticleGun*  fParticleGun; // G4 particle gun
};

#endif
```

V0RunAction.hh

```
#ifndef V0RunAction_h
#define V0RunAction_h 1

#include "G4UserRunAction.hh"
#include "globals.hh"

class G4Run;

class V0RunAction : public G4UserRunAction
{
  public:
    V0RunAction();
    virtual ~V0RunAction();

    virtual void BeginOfRunAction(const G4Run*);
    virtual void   EndOfRunAction(const G4Run*);
};

#endif
```

V0SteppingAction.hh

```
#ifndef V0SteppingAction_h
#define V0SteppingAction_h 1

#include "G4UserSteppingAction.hh"

class V0DetectorConstruction;
class V0EventAction;

class V0SteppingAction : public G4UserSteppingAction
{
public:
  V0SteppingAction(const V0DetectorConstruction* detectorConstruction,
                   V0EventAction* eventAction);
  virtual ~V0SteppingAction();

  virtual void UserSteppingAction(const G4Step* step);

private:
  const V0DetectorConstruction* fDetConstruction;
```

```
  V0EventAction*  fEventAction;
};

#endif
```

A.2.2 Clases Usuario

V0ActionInitialization.cc

```
#include "V0ActionInitialization.hh"
#include "V0PrimaryGeneratorAction.hh"
#include "V0RunAction.hh"
#include "V0EventAction.hh"
#include "V0SteppingAction.hh"
#include "V0DetectorConstruction.hh"

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......

V0ActionInitialization::V0ActionInitialization
                           (V0DetectorConstruction* detConstruction)
 : G4VUserActionInitialization(),
   fDetConstruction(detConstruction)
{}

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......

V0ActionInitialization::~V0ActionInitialization()
{}

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......

void V0ActionInitialization::BuildForMaster() const
{
  SetUserAction(new V0RunAction);
}

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......

void V0ActionInitialization::Build() const
{
  SetUserAction(new V0PrimaryGeneratorAction);
  SetUserAction(new V0RunAction);
  V0EventAction* eventAction = new V0EventAction;
  SetUserAction(eventAction);
  SetUserAction(new V0SteppingAction(fDetConstruction,eventAction));
}
```

V0DetectorConstruction.cc

```
#include "V0DetectorConstruction.hh"

#include "G4NistManager.hh"
#include "G4Box.hh"
```

```
#include "G4Trap.hh"
#include "G4UnionSolid.hh"
#include "G4LogicalVolume.hh"
#include "G4PVPlacement.hh"
#include "G4SystemOfUnits.hh"
#include "G4Material.hh"
#include "G4PVReplica.hh"
#include "G4AutoDelete.hh"
#include "G4GeometryManager.hh"
#include "G4PhysicalVolumeStore.hh"
#include "G4LogicalVolumeStore.hh"
#include "G4SolidStore.hh"
#include "G4VisAttributes.hh"
#include "G4Colour.hh"
#include "G4PhysicalConstants.hh"

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......

V0DetectorConstruction::V0DetectorConstruction()
: G4VUserDetectorConstruction(),
  fAbsPsX(0),
  fAbsPsX1(0),
  fAbsPsY(0),
  fAbsPsY1(0)
{ }

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......

V0DetectorConstruction::~V0DetectorConstruction()
{ }

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......

G4VPhysicalVolume* V0DetectorConstruction::Construct()
{
  // Get nist material manager
  G4NistManager* nistManager = G4NistManager::Instance();

  // Build materials
  // Lead material defined using NIST Manager
  nistManager->FindOrBuildMaterial("G4_Galactic");
  nistManager->FindOrBuildMaterial("G4_PLASTIC_SC_VINYLTOLUENE");

  // Geometry parameters
  G4int nofLayers = 21;
  G4int noLayersXY = 10; // numero de capas de barras orientadas en x e y
  G4double pDx1 = 19*mm;
  G4double pDx2 = pDx1;
  G4double pDy1 = nofLayers*pDx1+pDx1/2;
  G4double pDy2 = pDy1;
  G4double pDx3 = 0.000000000000000001*mm;
  G4double pDx4 = pDx3;
  G4double pDz = (sqrt(3)/2)*pDx1;
  G4double pTheta = 0*deg;
  G4double pDphi = 0*deg;
  G4double pAlph1 = 0*deg;
  G4double pAlph2 = 0*deg;

  G4double worldSizeX = 5*m;
  G4double worldSizeY = 5*m;
  G4double worldSizeZ = 1.5*noLayersXY*m;

  // Get materials
```

```
  G4Material* air = G4Material::GetMaterial("G4_Galactic");
  G4Material* plasticsc = G4Material::GetMaterial("G4_PLASTIC_SC_VINYLTOLUENE");

  //
  // World
  //

  G4VSolid* worldS
    = new G4Box("World",           // its name
                worldSizeX/2, worldSizeY/2, worldSizeZ/2); // its size

  G4LogicalVolume* worldLV
    = new G4LogicalVolume(
                worldS,           // its solid
                air,              // its material
                "World");         // its name

  G4VPhysicalVolume* worldPV
    = new G4PVPlacement(
                0,                // no rotation
                G4ThreeVector(),  // at (0,0,0)
                worldLV,          // its logical volume
                "World",          // its name
                0,                // its mother  volume
                false,            // no boolean operation
                0,                // copy number
                true);  // checking overlaps

//--------------------------------------------------------------------------

  //Bars

  G4VSolid* calorS
   = new G4Trap("calorS",
                 pDz/2, 30*deg, pDphi, pDy1/2, nofLayers*(pDx1/2+pDx3/2), nofLayers*(pDx2/2+pDx4/2), pAlph1, pDy2/2,
                       (nofLayers)*(pDx1/2+pDx3/2),(nofLayers)*(pDx2/2+pDx4/2), pAlph2);

   G4LogicalVolume* calorLV
   = new G4LogicalVolume(
                calorS,            // its solid
                air,               // its material
                "calorLV");        // its name

   for(G4int i = 0; i < noLayersXY; i++){
   new G4PVPlacement(
                0,                          // no rotation
                G4ThreeVector(0,0,2*i*pDz), // its position
                calorLV,                    // its logical volume
                "calor",                    // its name
                worldLV,                    // its mother  volume
                false,                      // no boolean operation
                0,                          // copy number
                true);                      // checking overlaps
}

  G4VSolid* trapS
    = new G4Trap("trap",
                 pDz/2, 30*deg, pDphi, pDy1/2, pDx1/2+pDx3/2, pDx2/2+pDx4/2, pAlph1, pDy2/2, pDx1/2+pDx3/2, pDx2/2+
                       pDx4/2, pAlph2);

  G4LogicalVolume* trapLV
    = new G4LogicalVolume(
                trapS,           // its solid
```

```
                air,            // its material
                "trap");        // its name

  new G4PVReplica(
                "trap",          // its name
                trapLV,          // its logical volume
                calorLV,         // its mother
                kXAxis,          // axis of replication
                nofLayers,       // number of replica
                (pDx1+pDx3));    // witdth of replica

  G4VSolid* trap2S
    = new G4Trap("trap2",
                 pDz/2, pTheta, pDphi, pDy1/2, pDx1/2, pDx2/2, pAlph1, pDy2/2, pDx3/2, pDx4/2, pAlph2);

  G4LogicalVolume* trap2LV
    = new G4LogicalVolume(
                trap2S,            // its solid
                plasticsc,         // its material
                "trap2");          // its name

   fAbsPsY
   = new G4PVPlacement(
                0,                        // no rotation
                G4ThreeVector(-pDx1/4,0,0), // its position
                trap2LV,                  // its logical volume
                "trap2",                  // its name
                trapLV,                   // its mother  volume
                false,                    // no boolean operation
                0,                        // copy number
                true);                    // checking overlaps

   G4VSolid* trap3S
    = new G4Trap("trap3",
                 pDz/2, pTheta, pDphi, pDy1/2, pDx3/2, pDx4/2, pAlph1, pDy2/2, pDx1/2, pDx2/2, pAlph2);

  G4LogicalVolume* trap3LV
    = new G4LogicalVolume(
                trap3S,            // its solid
                plasticsc,         // its material
                "trap3");          // its name

   fAbsPsY1
   = new G4PVPlacement(
                0,                        // no rotation
                G4ThreeVector(pDx1/4,0,0), // its position
                trap3LV,                  // its logical volume
                "trap3",                  // its name
                trapLV,                   // its mother  volume
                false,                    // no boolean operation
                0,                        // copy number
                true);                    // checking overlaps

//-----------------------------------------------------------------------------

   G4VSolid* calor2S
   = new G4Trap("calor2S",
                pDz/2, 30*deg, pDphi, pDy1/2, nofLayers*(pDx1/2+pDx3/2), nofLayers*(pDx2/2+pDx4/2), pAlph1, pDy2/2,
                      (nofLayers)*(pDx1/2+pDx3/2),(nofLayers)*(pDx2/2+pDx4/2), pAlph2);

   G4LogicalVolume* calor2LV
   = new G4LogicalVolume(
                calor2S,              // its solid
```

```
                air,                    // its material
                "calor2LV");            // its name

   G4RotationMatrix* rm = new G4RotationMatrix();

   rm->rotateZ(90*deg);

   for(G4int i = 0; i < noLayersXY; i++){
   new G4PVPlacement(
                rm,                             // no rotation
                G4ThreeVector(0,0,(2*i+1)*pDz), // its position
                calor2LV,                       // its logical volume
                "calor2",                       // its name
                worldLV,                        // its mother  volume
                false,                          // no boolean operation
                0,                              // copy number
                true);                          // checking overlaps
}


  G4VSolid* trap4S
    = new G4Trap("trap4",
                 pDz/2, 30*deg, pDphi, pDy1/2, pDx1/2+pDx3/2, pDx2/2+pDx4/2, pAlph1, pDy2/2, pDx1/2+pDx3/2, pDx2/2+
                       pDx4/2, pAlph2);

  G4LogicalVolume* trap4LV
    = new G4LogicalVolume(
                trap4S,           // its solid
                air,              // its material
                "trap4");         // its name

  new G4PVReplica(
                "trap4",          // its name
                trap4LV,          // its logical volume
                calor2LV,         // its mother
                kXAxis,           // axis of replication
                nofLayers,        // number of replica
                (pDx1+pDx3));     // witdth of replica


  G4VSolid* trap5S
    = new G4Trap("trap5",
                 pDz/2, pTheta, pDphi, pDy1/2, pDx1/2, pDx2/2, pAlph1, pDy2/2, pDx3/2, pDx4/2, pAlph2);

  G4LogicalVolume* trap5LV
    = new G4LogicalVolume(
                trap5S,             // its solid
                plasticsc,          // its material
                "trap5");           // its name

   fAbsPsX
   = new G4PVPlacement(
                0,                          // no rotation
                G4ThreeVector(-pDx1/4,0,0), // its position
                trap5LV,                    // its logical volume
                "trap5",                    // its name
                trap4LV,                    // its mother  volume
                false,                      // no boolean operation
                0,                          // copy number
                true);                      // checking overlaps

   G4VSolid* trap6S
    = new G4Trap("trap6",
```

```
                pDz/2, pTheta, pDphi, pDy1/2, pDx3/2, pDx4/2, pAlph1, pDy2/2, pDx1/2, pDx2/2, pAlph2);

  G4LogicalVolume* trap6LV
    = new G4LogicalVolume(
                trap6S,             // its solid
                plasticsc,          // its material
                "trap6");           // its name

   fAbsPsX1
   = new G4PVPlacement(
                0,                        // no rotation
                G4ThreeVector(pDx1/4,0,0), // its position
                trap6LV,                  // its logical volume
                "trap3",                  // its name
                trap4LV,                  // its mother  volume
                false,                    // no boolean operation
                0,                        // copy number
                true);  // checking overlaps

//------------------------------------------------------------------------

   worldLV->SetVisAttributes (G4VisAttributes::Invisible);

   G4VisAttributes* visAttributes;
   visAttributes = new G4VisAttributes(G4Colour(0,0,1.0));
   trap5LV->SetVisAttributes(visAttributes);

   G4VisAttributes* visAttributes1;
   visAttributes1 = new G4VisAttributes(G4Colour(1.0,0,0));
   trap6LV->SetVisAttributes(visAttributes1);


   G4VisAttributes* visAttributes2;
   visAttributes2 = new G4VisAttributes(G4Colour(0,1.0,0));
   trap2LV->SetVisAttributes(visAttributes2);

   G4VisAttributes* visAttributes3;
   visAttributes3 = new G4VisAttributes(G4Colour(1.0,1.0,0));
   trap3LV->SetVisAttributes(visAttributes3);

  return worldPV;
}
```

V0EventAction.cc

```
#include "V0EventAction.hh"

#include "V0RunAction.hh"
#include "V0Analysis.hh"
#include "G4RunManager.hh"
#include "G4Event.hh"
#include "G4UnitsTable.hh"
#include "Randomize.hh"
#include <iomanip>

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......

V0EventAction::V0EventAction()
 : G4UserEventAction(),
    fEnergyAbsPsX(0),
```

```
    fEnergyAbsPsX1(0),
    fEnergyAbsPsY(0),
    fEnergyAbsPsY1(0),
    fTrackLAbsPsX(0),
    fTrackLAbsPsX1(0),
    fTrackLAbsPsY(0),
    fTrackLAbsPsY1(0),
    fPosX(0),
    fPosY(0)


{}

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......

V0EventAction::~V0EventAction()
{}

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......

void V0EventAction::BeginOfEventAction(const G4Event* /*event*/)
{
  // initialisation per event
    fEnergyAbsPsX = 0.;
    fEnergyAbsPsX1 = 0.;
    fEnergyAbsPsY = 0.;
    fEnergyAbsPsY1 = 0.;
    fTrackLAbsPsX = 0.;
    fTrackLAbsPsX1 = 0.;
    fTrackLAbsPsY = 0.;
    fTrackLAbsPsY1 = 0.;
    fPosX = 0;
    fPosY = 0;


}

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......

void V0EventAction::EndOfEventAction(const G4Event* event)
{
  // Accumulate statistics
  //

  // get analysis manager
  G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();

  // fill histograms
  analysisManager->FillH1(1, fEnergyAbsPsX + fEnergyAbsPsX1 + fEnergyAbsPsY + fEnergyAbsPsY1);
  analysisManager->FillH1(2, fTrackLAbsPsX + fTrackLAbsPsX1 + fTrackLAbsPsY + fTrackLAbsPsY1);
  analysisManager->FillH1(3, fPosX);
  analysisManager->FillH1(4, fPosY);
  analysisManager->FillH2(1, fPosX, fPosY);


  // Print per event (modulo n)
  //
  G4int eventID = event->GetEventID();
  G4int printModulo = G4RunManager::GetRunManager()->GetPrintProgress();
  if ( ( printModulo > 0 ) && ( eventID % printModulo == 0 ) ) {
    G4cout << "---> End of event: " << eventID << G4endl;

    G4cout
```

```
      << "   Total energy: " << std::setw(7)
                                    << G4BestUnit(fEnergyAbsPsX + fEnergyAbsPsX1 + fEnergyAbsPsY + fEnergyAbsPsY1,
                                          "Energy")
      << "   Total track length: " << std::setw(7)
                                    << G4BestUnit(fTrackLAbsPsX + fTrackLAbsPsX1 + fTrackLAbsPsY + fTrackLAbsPsY1,
                                          "Length")
      << G4endl;
  }
}
```

V0PrimaryGeneratorAction.cc

```
#include "V0PrimaryGeneratorAction.hh"

#include "G4RunManager.hh"
#include "G4LogicalVolumeStore.hh"
#include "G4LogicalVolume.hh"
#include "G4Box.hh"
#include "G4Event.hh"
#include "G4ParticleGun.hh"
#include "G4ParticleTable.hh"
#include "G4ParticleDefinition.hh"
#include "G4SystemOfUnits.hh"
#include "Randomize.hh"
#include "G4GeneralParticleSource.hh"

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......

V0PrimaryGeneratorAction::V0PrimaryGeneratorAction()
 : G4VUserPrimaryGeneratorAction(),
   fParticleGun(0)
{
  G4int nofParticles = 1;
  fParticleGun = new G4ParticleGun(nofParticles);

  // default particle kinematic
  //
  G4ParticleDefinition* particleDefinition
    = G4ParticleTable::GetParticleTable()->FindParticle("mu-");
  fParticleGun->SetParticleDefinition(particleDefinition);
  fParticleGun->SetParticleMomentumDirection(G4ThreeVector(0.,0.,1.));
  fParticleGun->SetParticleEnergy(2.*GeV);
}

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......

V0PrimaryGeneratorAction::~V0PrimaryGeneratorAction()
{
  delete fParticleGun;
}

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......

void V0PrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{

  G4double worldZHalfLength = 0;
  G4LogicalVolume* worlLV
    = G4LogicalVolumeStore::GetInstance()->GetVolume("World");
```

```
  G4Box* worldBox = 0;
  if ( worlLV) worldBox = dynamic_cast< G4Box*>(worlLV->GetSolid());
  if ( worldBox ) {
    worldZHalfLength = worldBox->GetZHalfLength();
  }
  else  {
    G4ExceptionDescription msg;
    msg << "World volume of box not found." << G4endl;
    msg << "Perhaps you have changed geometry." << G4endl;
    msg << "The gun will be place in the center.";
    G4Exception("V0PrimaryGeneratorAction::GeneratePrimaries()",
      "MyCode0002", JustWarning, msg);
  }

  // Set gun position
  fParticleGun
    ->SetParticlePosition(G4ThreeVector(0, 0, -3*worldZHalfLength/4));

   if (1)
    {
      G4ThreeVector oldPosition = fParticleGun->GetParticlePosition();
      G4double x0 = oldPosition.x() + (2*G4UniformRand()-1.)*20*cm;
      G4double y0 = oldPosition.y() + (2*G4UniformRand()-1.)*20*cm;
      fParticleGun->SetParticlePosition(G4ThreeVector(x0,y0,-3*worldZHalfLength/4));
      fParticleGun->GeneratePrimaryVertex(anEvent);
      fParticleGun->SetParticlePosition(oldPosition);
    }
  else  fParticleGun->GeneratePrimaryVertex(anEvent);
}
```

V0RunAction.cc

```
#include "V0RunAction.hh"
#include "V0Analysis.hh"

#include "G4Run.hh"
#include "G4RunManager.hh"
#include "G4UnitsTable.hh"
#include "G4SystemOfUnits.hh"

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......

V0RunAction::V0RunAction()
 : G4UserRunAction()
{
  // set printing event number per each event
  G4RunManager::GetRunManager()->SetPrintProgress(1);

  // Create analysis manager
  // The choice of analysis technology is done via selectin of a namespace
  // in V0Analysis.hh
  G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();
  G4cout << "Using " << analysisManager->GetType() << G4endl;

  // Create directories

  analysisManager->SetVerboseLevel(1);
  analysisManager->SetFirstHistoId(1);

  // Creating histograms
```

```
  analysisManager->CreateH1("1","Edep in absorber", 140, 0., 300*MeV);
  analysisManager->CreateH1("2","trackL in absorber", 140, 0., 1*m);
  analysisManager->CreateH1("3","PosX", 140, -30*cm, 30*cm);
  analysisManager->CreateH1("4","PosY", 140, -30*cm, 30*cm);
  analysisManager->CreateH2("5","PosXY", 140, -30*cm, 30*cm, 140, -30*cm, 30*cm);
  analysisManager->CreateH2("6","PosXY", 140, -30*cm, 30*cm, 140, -30*cm, 30*cm);
  analysisManager->CreateH2("7","PosXZ", 140, -1*m, 1*m, 140, -1*m, 1*m);
  analysisManager->CreateH2("8","PosYZ", 140, -1*m, 1*m, 140, -1*m, 1*m);
  analysisManager->CreateH3("9","PosXYZ", 140, -40*cm, 40*cm, 140, -40*cm, 40*cm, 140, -40*cm, 40*cm);

  // Creating ntuple
  //
  analysisManager->CreateNtuple("V0", "Edep and TrackL");
  analysisManager->CreateNtupleSColumn("Process");
  analysisManager->CreateNtupleSColumn("Particle");
  analysisManager->FinishNtuple();
}

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......

V0RunAction::~V0RunAction()
{
  delete G4AnalysisManager::Instance();
}

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......

void V0RunAction::BeginOfRunAction(const G4Run* /*run*/)
{
  //inform the runManager to save random number seed
  //G4RunManager::GetRunManager()->SetRandomNumberStore(true);

  // Get analysis manager
  G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();

  // Open an output file
  //
  G4String fileName = "V0";
  analysisManager->OpenFile(fileName);
}

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......

void V0RunAction::EndOfRunAction(const G4Run* /*run*/)
{
  // print histogram statistics
  //
  G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();
  if ( analysisManager->GetH1(1) ) {
    G4cout << G4endl << " ----> print histograms statistic ";
    if(isMaster) {
      G4cout << "for the entire run " << G4endl << G4endl;
    }
    else {
      G4cout << "for the local thread " << G4endl << G4endl;
    }

    G4cout << " EAbs : mean = "
       << G4BestUnit(analysisManager->GetH1(1)->mean(), "Energy")
       << " rms = "
       << G4BestUnit(analysisManager->GetH1(1)->rms(),  "Energy") << G4endl;

    G4cout << " LAbs : mean = "
```

```
      << G4BestUnit(analysisManager->GetH1(2)->mean(), "Length")
      << " rms = "
      << G4BestUnit(analysisManager->GetH1(2)->rms(),  "Length") << G4endl;

  }

  // save histograms & ntuple
  //
  analysisManager->Write();
  analysisManager->CloseFile();
}
```

V0SteppingAction.cc

```
#include "V0SteppingAction.hh"
#include "V0EventAction.hh"
#include "V0DetectorConstruction.hh"
#include "V0Analysis.hh"

#include "G4SystemOfUnits.hh"
#include "G4Step.hh"
#include "G4RunManager.hh"

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......

V0SteppingAction::V0SteppingAction(
                    const V0DetectorConstruction* detectorConstruction,
                    V0EventAction* eventAction)
  : G4UserSteppingAction(),
    fDetConstruction(detectorConstruction),
    fEventAction(eventAction)
{ }

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......

V0SteppingAction::~V0SteppingAction()
{ }

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......

void V0SteppingAction::UserSteppingAction(const G4Step* step)
{
// Collect energy and track length step by step

  G4String procName = step->GetPostStepPoint()->GetProcessDefinedStep()->GetProcessName();
  G4String particleName = step->GetTrack()->GetDynamicParticle()->GetDefinition()->GetParticleName();
  G4StepPoint* prepoint = step->GetPreStepPoint();
  G4StepPoint* postpoint = step->GetPostStepPoint();
  G4ThreeVector pos1 = prepoint->GetPosition();
  G4ThreeVector pos2 = postpoint->GetPosition();
  G4double posX = pos1.x();
  G4double posY = pos1.y();
  G4double posZ = pos1.z();

  // get volume of the current step
  G4VPhysicalVolume* volume = step->GetPreStepPoint()->GetTouchableHandle()->GetVolume();

  // energy deposit
  G4double edep = step->GetTotalEnergyDeposit();
```

```
  // step length
  G4double stepLength = 0.;

  G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();

  if ( step->GetTrack()->GetDefinition()->GetPDGCharge() != 0. ) {
    stepLength = step->GetStepLength();
  }

  if ( volume == fDetConstruction->GetAbsPsX() ) {
    fEventAction->AddAbsPsX(edep,stepLength);
  }

  if ( volume == fDetConstruction->GetAbsPsX1() ) {
    fEventAction->AddAbsPsX1(edep,stepLength);
  }

  if ( volume == fDetConstruction->GetAbsPsY() ) {
    fEventAction->AddAbsPsY(edep,stepLength);
    fEventAction->AddPosX(pos1.x());
    fEventAction->AddPosY(pos1.y());
  }

  if ( volume == fDetConstruction->GetAbsPsY1() ) {
    fEventAction->AddAbsPsY1(edep,stepLength);
    fEventAction->AddPosX(pos1.x());
    fEventAction->AddPosY(pos1.y());
  }

  if ( volume == fDetConstruction->GetAbsPsX() || volume == fDetConstruction->GetAbsPsX1() || volume ==
        fDetConstruction->GetAbsPsY() ||volume == fDetConstruction->GetAbsPsY1()) {
    analysisManager->FillH3(1,posZ,posY,posX);
    analysisManager->FillH2(2,posX,posY);
    analysisManager->FillH2(3,posX,posZ);
    analysisManager->FillH2(4,posY,posZ);
    }

    analysisManager->FillNtupleSColumn(0,procName);
    analysisManager->FillNtupleSColumn(1,particleName);
    analysisManager->AddNtupleRow();

}
```

detectorV0.cc

```
#include "V0DetectorConstruction.hh"
#include "V0ActionInitialization.hh"

#ifdef G4MULTITHREADED
#include "G4MTRunManager.hh"
#else
#include "G4RunManager.hh"
#endif

#include "G4UImanager.hh"
#include "G4UIcommand.hh"
#include "FTFP_BERT.hh"
#include "Randomize.hh"
```

```
#include "G4VisExecutive.hh"
#include "G4UIExecutive.hh"

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......

namespace {
  void PrintUsage() {
    G4cerr << " Usage: " << G4endl;
    G4cerr << " exampleV0 [-m macro ] [-u UIsession] [-t nThreads]" << G4endl;
    G4cerr << "   note: -t option is available only for multi-threaded mode."
           << G4endl;
  }
}

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......

int main(int argc,char** argv)
{
  // Evaluate arguments
  //
  if ( argc > 7 ) {
    PrintUsage();
    return 1;
  }

  G4String macro;
  G4String session;
#ifdef G4MULTITHREADED
  G4int nThreads = 0;
#endif
  for ( G4int i=1; i<argc; i=i+2 ) {
    if      ( G4String(argv[i]) == "-m" ) macro = argv[i+1];
    else if ( G4String(argv[i]) == "-u" ) session = argv[i+1];
#ifdef G4MULTITHREADED
    else if ( G4String(argv[i]) == "-t" ) {
      nThreads = G4UIcommand::ConvertToInt(argv[i+1]);
    }
#endif
    else {
      PrintUsage();
      return 1;
    }
  }

  // Detect interactive mode (if no macro provided) and define UI session
  //
  G4UIExecutive* ui = 0;
  if ( ! macro.size() ) {
    ui = new G4UIExecutive(argc, argv);
  }

  // Choose the Random engine
  //
  G4Random::setTheEngine(new CLHEP::RanecuEngine);

  //Random runs
  //-------------------------------------------------------------------------------
  long seeds[2];
  time_t systime = time(NULL);
 seeds[0] = (long) systime;
 seeds[1] = (long) (systime * G4UniformRand());
  G4Random::setTheSeeds(seeds);
  //-------------------------------------------------------------------------------
```

```
  // Construct the default run manager
  //
#ifdef G4MULTITHREADED
  G4MTRunManager * runManager = new G4MTRunManager;
  if ( nThreads > 0 ) {
    runManager->SetNumberOfThreads(nThreads);
  }
#else
  G4RunManager * runManager = new G4RunManager;
#endif

  // Set mandatory initialization classes
  //
  V0DetectorConstruction* detConstruction = new V0DetectorConstruction();
  runManager->SetUserInitialization(detConstruction);

  G4VModularPhysicsList* physicsList = new FTFP_BERT;
  runManager->SetUserInitialization(physicsList);

  V0ActionInitialization* actionInitialization
     = new V0ActionInitialization(detConstruction);
  runManager->SetUserInitialization(actionInitialization);

  // Initialize visualization
  //
  G4VisManager* visManager = new G4VisExecutive;
  visManager->Initialize();

  // Get the pointer to the User Interface manager
  G4UImanager* UImanager = G4UImanager::GetUIpointer();

  // Process macro or start UI session
  //
  if ( macro.size() ) {
    // batch mode
    G4String command = "/control/execute ";
    UImanager->ApplyCommand(command+macro);
  }
  else  {
    // interactive mode : define UI session
    UImanager->ApplyCommand("/control/execute init_vis.mac");
    if (ui->IsGUI()) {
       UImanager->ApplyCommand("/control/execute icons.mac");
    }
    ui->SessionStart();
    delete ui;
  }

  delete visManager;
  delete runManager;
}
```

Bibliografía

[1] D. HECK ET AL, CORSIKA: A Monte Carlo Code to Simulate Extensive Air Showers, CORSIKA 7.4004 .

[2] J. ALLISON ET AL, Geant4 Developments and Applications, IEEE Transactions on Nuclear Science 53 No. 1, (2006) 270-278.

[3] S. AGOSTINELLI ET AL, Geant4 - A Simulation Toolkit, Nuclear Instruments and Methods A 506, (2003) 250-303.

[4] R. A. MILLIKAN AND G. H. CAMERON. *High frequency rays of cosmic origin iii measurements in snow-fed lakes at high altitudes. Physical Review, 28(5):851-868* 1926.

[5] T. WULF. *Physikalische Zeitschrift 11, 811.* 1910.

[6] V. HESS. *Physikalische Zeitschrschift 13, 1084.* 1912.

[7] W. BOTHE AND W. KOLHÖRSTER. *Zeitschrift für Physik 56, 751.* 1929.

[8] PIERRE AUGER, P. EHRENFEST, R. MAZE, J. DAUDIN, AND ROBLEY A. FRÉON. *Extensive Cosmic-Ray Showers. Rev. Mod. Phys. 11, 288.* 1939.

[9] G. W. CLARK, J. EARL, W. L. KRAUSHAAR, J. LINSLEY, B. B. ROSSI, F. SCHERB, AND D. W. SCOTT. *Cosmic-Ray Air Showers at Sea Level. Phys. Rev. 122, 637.* 1961.

[10] JOHN LINSLEY. *Evidence for a Primary Cosmic-Ray Particle with Energy* 10^{20} *eV. Phys. Rev. Lett. 10, 146.* 1963.

[11] D.J. BIRD ET AL. *[Fly's Eye Collaboration], Phys. Rev. Lett. 71, 3401* 1993.

[12] K. OLIVE ET AL. *Particle Data Group. Chin.Phys. C38, 090001.* 2014.

[13] THOMAS K. GAISSER, TODOR STANEV. *High-energy Cosmic Rays. arXiv:astro-ph/0510321.* 2005.

[14] M. S. LONGAIR. *High Energy Astrophysics. Cambridge Univ. Press.* 1981.

[15] COMPARISON OF HADRONIC INTERACTION MODELS AT AUGER ENERGIES. *D. Heck, M. Risse, J. Knapp. arXiv:astro-ph/0210392* 2002.

[16] K. GREISEN. *Ann. Rev. Nucl. Sci. 10* (1960) 63.

[17] K. KAMATA, J. NISHIMURA. *Prog. Theoret. Phys. Suppl. 6* (1958) 93.

[18] SYED NAEEM AHMED. *Physics and Engineering of Radiation Detection.* pag. 337,339,353 - 2007

[19] RENE BRUN AND FONS RADEMAKERS. *ROOT - An Object Oriented Data Analysis Framework, Proceedings AIHENP'96 Workshop, Lausanne,* Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. A 389 (1997) 81-86. See also http://root.cern.ch/.

[20] L. ALIAGA, L. BAGBY, D.A. MARTINEZ CAICEDO ET AL, Design, Calibration, and Performance of the MINERvA Detector, *arXiv:1305.5199 [physics.ins-det]* 2013

[21] A TAPIA ET AL. *Study of the chemical composition of high energy cosmic rays using the muon LDF of EAS between* $10^{17.25}$ *eV and* $10^{17.50}$ *eV, arXiv:1501.02217.*

[22] 38TH ICHEP POSTER CONTRIBUTION. *https://indico.cern.ch/event/432527/contributions/1071873/.*