Universidad de Nariño

Facultad de Ciencias Exactas y Naturales

Departamento de Física

Simulación de los fenómenos ópticos de propagación de las partículas dentro de un prototipo de detector de muones atmosféricos

TRABAJO DE GRADO

Para optar al título profesional de:

Físico

Jeferson Daniel Rosero Benavides

San Juan de Pasto, Colombia

Octubre 2022

Universidad de Nariño

Facultad de Ciencias Exactas y Naturales

Departamento de Física

Simulación de los fenómenos ópticos de propagación de las partículas dentro de un prototipo de detector de muones atmosféricos

Jeferson Daniel Rosero Benavides

TRABAJO DE GRADO

Director:

Alex Marcelo Tapia Casanova, PhD

Doctor en Física

San Juan de Pasto, Colombia

Octubre 2022

San Juan de Pasto, Octubre 2022

Nota de Aceptación

_____

_____

_____

_____

_____

_____

Director: Alex Marcelo Tapia Casanova, PhD

Jurado: Eduardo Rojas Peña, PhD

Jurado: Johana Herrera Ruales

San Juan de Pasto, Octubre 2022

Agradecimientos

A mis padres y a mi hermana por su apoyo incondicional durante toda mi vida académica.

Dedicado a toda mi familia y amigos, gracias por su animo y consideración.

Simulación de los fenómenos ópticos de propagación de las partículas dentro de un prototipo de detector de muones atmosféricos

Resumen

El paso de radiación a través de la materia donde ocurren distintos procesos producto de dicha interacción, nos proporciona un panorama importante a la hora de estudiar los procesos físicos que se desarrollan dentro de la materia. La interacción de muones atmosféricos, los cuales son producto de las lluvias atmosféricas extendidas, con un prototipo de detector construido con placas de centelleo plásticas que sera simulado usando la herramienta computacional GEANT4, con esto se pretende tener una propuesta del diseño final del detector de muones. Este proyecto tiene como objetivo estudiar los procesos ópticos que se desarrollan dentro de una placa plástica de centelleo durante el paso de muones atmosféricos. Las simulaciones realizadas y el respectivo análisis de datos, podrán servir como una guía importante para el diseño de un sistema tomográfico basado en muones atmosféricos generados por rayos cósmicos en el área de muongrafía.

Simulation of the optical phenomena of propagation of particles within a atmospheric muon detector prototype

Abstract

The passage of radiation through matter where different processes occur as a result of said interaction, provides us with an important overview when studying the physical processes that take place within matter. The interaction of atmospheric muons, which are the product of extended atmospheric showers, with a detector prototype built with plastic scintillation plates that will be simulated using the GEANT4 computational tool, with this it is intended to have a proposal for the final design of the muon detector. This project aims to study the optical processes that take place inside a plastic scintillation plate during the passage of atmospheric muons. The simulations carried out and the respective data analysis may serve as an important guide for the design of a tomographic system based on atmospheric muons generated by cosmic rays in the area of muonography.

Índice general

Glosario

Difusión (en reflexión): Es la reflexión de la luz desde una superficie, de tal forma que un rayo incidente es reflejado en muchos ángulos, en lugar de únicamente un solo ángulo, como en el caso de la reflexión especular.

Centellador: Es un material que centellea, o sea, exhibe luminiscencia cuando por él pasa radiación ionizante (electrones, positrones u otras partículas o iones más pesados). Esto se produce porque el material absorbe parte de la energía de la partícula incidente y la reemite en forma de un corto destello de luz, típicamente en el rango de la luz visible.

Luminiscencia: Es todo proceso de emisión de luz cuyo origen no radica exclusivamente en las altas temperaturas sino que, por el contrario, es una forma de ”luz fría” en la que la emisión de radiación lumínica es provocada en condiciones de temperatura ambiente o baja.

Espalación: Es el proceso en el cual un núcleo pesado emite un gran número de neutrones al ser golpeado por una partícula altamente energética, lo que resulta en una reducción drástica de su peso atómico.

Kilopársec (kpc): Es una unidad de longitud utilizada en astronomía. mil pársecs equivale a 3262 años luz.

Nucleosíntesis: Es el proceso de creación de nuevos núcleos atómicos a partir de los nucleones preexistentes (protones y neutrones) para llegar a generar el resto de los elementos de la tabla periódica.

Lambertiano: Es una superficie ideal que refleja la energía incidente desde una dirección igual en todas direcciones.

Semiconductor: Es un elemento que se comporta o bien como un conductor o bien como un aislante dependiendo de diversos factores, por ejemplo: el campo eléctrico o magnético, la presión, la radiación que le incide, o la temperatura del ambiente en el que se encuentre.

Capítulo 1

Introducción

La física experimental moderna tanto nuclear como de partículas nos proporciona un gran avance a la hora de crear modelos que nos proporcionan herramientas y aplicaciones para los retos que nos plantean. En particular se destaca la importancia, que a lo largo de los años ha incrementado, por la simulación precisa y completa de detectores de partículas. En este caso los detectores de centellador plástico, utilizados por su alto grado de resolución y sensibilidad. Al contar con estas características es recomendable su uso para la investigación de experimentos en entornos de muy alta energía, por lo que las partículas involucradas en dichos procesos serán de un tipo característico y cuyo nacimiento se ve reflejado en lo que se conoce como los rayos cósmicos.

Nuestro interés de investigación se va a centrar en los muones creados en la atmósfera producto de dichas lluvias atmosféricas generadas por rayos cósmicos; vale la pena recordar que el muon es una partícula similar al electrón pero con una contribución de masa mucho más grande. La detección de de dichas partículas puede ser registrada a partir de un detector construido utilizando material centellador plástico, pues se beneficia de las propiedades de este material para emitir fotones cuando los átomos son excitados debido al paso de la radiación ionizante, se han adoptado detectores de este tipo en muchos detectores de partículas recientes como en los experimentos MINOS [1] y OPERA [2] por nombrar algunos ejemplos.

Un diseño básico de un detector de muones atmosféricos incluye las siguientes características: esta formado por una geometría de material centellador plástico con unas dimensiones especificas, esta a su vez esta acoplada a un sistema detector de fotones ópticos, como por ejemplo, fotomultiplicadores estándar (PMT por sus siglas en ingles) o de silicio (SiMP por sus siglas en ingles). En ocasiones también se suele añadir un sistema de fibra de desplazamiento de longitud de onda (WLS por sus siglas en ingles), esto es con el fin de disminuir el efecto de autoabsorcion de fotones en el detector de centellador plástico.

En este trabajo se presenta una propuesta para llegar a construir un sistema tomográfico basado en muones atmosféricos. La tomografía de muones estudia la dispersión de estas partículas que depende en gran medida del número atómico del material que se desea atravesar, de esta forma se puede llegar a conocer su posición así como procesos de energía involucrados en su recorrido. Dicho procedimiento requiere de un seguimiento preciso y eficiente de los muones antes y después de atravesar el volumen sondeado. Es por eso que se estudia el diseño óptimo de un detector de centellador plástico, así como también se analiza un posible sistema de lectura. El alcance de este trabajo es realizar una simulación detallada de la respuesta de un dispositivo de detección de muones para ser utilizado en una posible prueba de rayos cósmicos. Para ello se hace uso de la herramienta computacional de simulación GEANT4 [3] la cuál nos permite rastrear explícitamente los fotones dentro de un medio especifico, teniendo en cuenta las propiedades ópticas que posee el detector de material centellador plástico, dichas propiedades son: La emisión, la absorción, la reflexión, la refracción, la difusión, entre otras. Resulta luego apropiado describir de manera realista la respuesta de los sistemas de centelleo dentro del enfoque de simulación de GEANT4.

Capítulo 2

Rayos cósmicos

2.1. Antecedentes

Los rayos cósmicos son partículas subatómicas que impactan la atmósfera de la tierra en un radio de aproximadamente 1000 eventos por metro cuadrado por segundo. Estos son núcleos ionizados, cerca de 90 % protones, 9 % de partículas alfa y el resto se convierten en núcleos pesados, y son distinguibles por sus altas energías que son comparables o incluso algo mayor que su propia masa. Sin embargo, existen algunos rayos cósmicos que tienen energías ultrarelativistas las cuales están por encima de los $10^{20}eV$ (al rededor de $20julios$), lo cual es once ordenes de magnitud más grande que el equivalente a la energía de la masa en reposo de un protón. Algunas de las cuestiones fundamentales para la física moderna son; ¿De donde provienen estos rayos cósmicos? y en particular ¿Como es que son acelerados a tan altas energías?. Las respuestas a estas preguntas sobre los rayos cósmicos son aun desconocidas, esta claro, además, que la mayoría de ellos llega desde fuera del sistema solar y desde el interior de nuestra galaxia a excepción de los rayos cósmicos de muy alta energía que deben tener su origen en lugares donde existan campos magnéticos mucho mayores que el de nuestra galaxia, a estos de lo denomina rayos cósmicos de origen extragalático.

El descubrimiento de los rayos cósmicos se remonta a finales del siglo XIX cuando científicos usando electroscopios para estudiar la conductividad de los gases, descubrieron que no importa con cuanto cuidado ellos aislaran sus electroscopios de posibles fuentes de radiación externa, estos todavía se descargaban a un ritmo lento. En 1901 dos grupos investigaron este fenómeno, J. Elster y H.Geitel en Alemania, y C.T. R. Wilson en Inglaterra. Ambos grupos concluyeron que alguna fuente desconocida de radiación ionizante estaba presente. Wilson

incluso sugirió que dicha ionización podría ser debido a la radiación de fuentes fuera de nuestra atmósfera, posiblemente radiación como los rayos Roentgen o como rayos catódicos, pero de mucho mayor poder de penetración. Un año después dos grupos en Canadá, Ernst Rutherford y H. Lester Cooke en la Universidad McGill y J. C. McLennan y E.F.Burton, en la Universidad de Toronto demostraron que $5cm$ de plomo redujeron esta misteriosa radiación en un $30\,\%$. En 1907 el padre Theodore Wulf del Instituto de Física de Ignatus, Universidad en Valkenburg, Holanda, inventó un nuevo electroscopio. El electroscopio de Wulf permitió a los científicos llevar la búsqueda del origen de la misteriosa radiación del laboratorio a las montañas, en lo alto de la Torre Eiffel y, finalmente, en globos. Suponiendo que la radiación provenía de la Tierra, esperaban encontrar una rápida disminución de la radiación a medida que se alejaban de la superficie. Sin en bargo, no encontraron tal disminución que esperaban y en algunos casos parecía haber pruebas de que la radiación en realidad había aumentado. Intrigado por los resultados contradictorios obtenidos por Wulf y sus colegas, un joven físico nuclear austríaco, Viktor Hess, obtuvo apoyo de la Academia Imperial de Ciencias de Austria y la Royal Austrian Aero Club para realizar una serie de vuelos en globo para estudiar tal radiación. Hess obtuvo una licencia para pilotar globos con el fin reducir el tamaño de la tripulación y por lo tanto aumentar la altitud a la que podía llevar sus electroscopios. El 12 de agosto de 1912, utilizando el Böhmen lleno de hidrógeno, Hess alcanzó una altitud de $5350m$. Llevaba dos cámaras de iones herméticamente selladas, encontró que la tasa de ionización inicialmente disminuyó, pero que a eso de los $1500m$ empezó a subir, hasta que a los $5000m$ acabó deteniéndose en el doble de la tasa inicial. Hess concluyó que los resultados de estas observaciones podrían explicarse mejor por la suposición de que existe una radiación de un poder de penetración muy alto desde arriba que entra en la atmósfera y causa parcialmente, incluso en las capas atmosféricas inferiores, ionización en los instrumentos de medición. El espectro no térmico de los rayos cósmicos indica que su aceleración es el resultado de procesos estocásticos en presencia de campos magnéticos, como propuso por primera vez a mediados de 1949 Fermi [4]. La aceleración difusiva de partículas cargadas en las ondas de choque de las explosiones de supernova puede dar origen a la mayoría de los rayos cósmicos, pero las energías más altas de los rayos cósmicos son tan extremas que no

se comprende si un mecanismo similar puede acelerar estas partículas incluso en las fuentes más extremas, que pueden o no ser lo suficientemente grandes y tener campos magnéticos suficientemente grandes.

Hace más de cuarenta años, en 1963, John Linsley publicó un artículo sobre la detección de un rayo cósmico de energía 10^{20} eV [5]. El artículo no pasó desapercibido, ni suscitó muchos comentarios. Los pocos físicos que fueron los interesados en los rayos cósmicos de alta energía se convencieron de que el espectro de energía de los rayos cósmicos puede continuar para siempre. El hecho de que los rayos cósmicos puedan tener energías superiores a 10^6 GeV (10^{15} eV) fue establecido a finales de los años treinta por Pierre Auger y sus colaboradores [6]. Mientras tanto, se detectaron lluvias de energía cada vez más alta alcanzando una lluvia de 10^{20} eV, luego entonces, parece una cuestión de tiempo y exposición. Ya en los años cincuenta se discutía sobre el origen de tales rayos cósmicos de ultra alta energía (UHECR, por sus siglas en ingles) y Cocconi llegó a la conclusión de que deben ser de origen extragaláctico ya que los rayos cósmicos galácticos no son lo suficientemente fuertes como para contener este tipo de partículas. Lo exclusivo de este evento se hizo evidente tres años después, tras el descubrimiento de la radiación cósmica del fondo de microondas [7] (CMBR, por sus siglas en ingles).

Como podemos ver la importancia de los rayos cósmicos juegan un papel crucial en la física de partículas pues su descubrimiento fue relevante a la hora de estudiar propiedades de partículas elementales involucradas en este tipo de fenómenos.

2.2. Composición de los rayos cósmicos

La relativa abundancia de rayos cósmicos son comparados con la abundancia de elementos en el sistema solar en la figura 2.1 se muestra lo siguiente: la linea azul muestra datos de baja energía, de $70 - 280 MeV/A$. Y la linea roja, muestra datos de alta energía, de $1000 - 2000 MeV/A$. Las partículas de rayos cósmicos son en su mayoría hidrógeno ($87\,\%$) y algo de helio ($12\,\%$) con cantidades decrecientes de carbono, oxígeno, etc. y de elementos más pesados. todos son completamente ionizado. Los electrones representan aproxima-

damente el $1\,\%$ de los rayos cósmicos. Con algunas excepciones, su composición química corresponde a la misma abundancia elemental que en nuestro sistema solar. Las excepciones (H, He son poco abundantes, Li, Be, B son demasiado abundantes) arrojan información importante sobre la materia atravesada por los rayos cósmicos. Abundancia isotópica de rayos cósmicos galácticos son muy similares a la composición isotópica del gas interestelar. Un excepción importante es la gran relación de $^{22}Ne/^{20}Ne$. La composición isotópica proporciona información clave sobre el origen, la aceleración y los mecanismos de transporte de rayos cósmicos en nuestra galaxia, lo que significa un gran avance para estudiar su naturaleza.

Veamos ahora 2 puntos importantes. el primero, núcleos con $Z > 1$ son mucho más abundantes en protones en los rayos cósmicos que en materiales del sistema solar. El por qué de este hecho no es del todo concreto, pero se podría deber al hecho de que el hidrógeno es relativamente ionizado más fuertemente por inyección en procesos de aceleración o por otro lado, podría deberse a diferencias genuinas en la composición de la fuente.
Por otro lado, el segundo punto a tener en cuenta es una herramienta importante para entender la propagación y el confinamiento de rayos cósmicos en la galaxia. Los dos grupos de elementos Li, Be, B y Sc, Ti, V, Cr, Mn se encuentran en ordenes de magnitud más abundantes en los rayos cósmicos que en los elementos del sistema solar, estos elementos están esencialmente ausentes como productos finales de la nucleosíntesis estelar. Ellos están a pesar de eso en la radiación cósmica como productos de la espalación de los abundantes núcleos de carbono y oxigeno (Li, Be, B) y de hierro (Sc, Ti, V, Cr, Mn). Estos son producto de colisiones de rayos cósmicos en el medio interestelar (ISM por sus siglas en ingles) [8]. Del conocimiento de la sección transversal por espalación, uno puede aprender algo acerca del material atravesado por rayos cósmicos por medio de producción y observación (cabe resaltar las implicaciones de partículas secundarias como fotones, neutrinos y antiprotones que serán producidos en cierta taza como rayos cósmicos propagándose a través de ISM). Para la mayoría de los rayos cósmicos la cantidad promedio de materia atravesada es del orden de $X = 5$ para una densidad de $10g/cm^3$, luego podemos calcular el grosor del material correspondiente a la distancia (l) de:

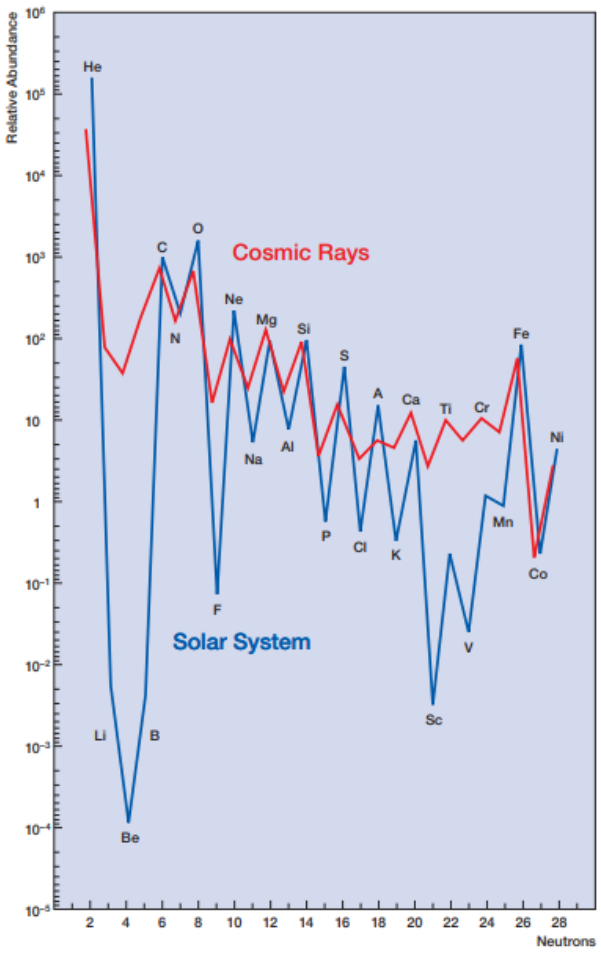

Figura 2.1: Abundancia relativa desde He hasta Ni de rayos cósmicos (linea roja), y de elementos en el sistema solar (linea azul)[8]

$$l = X/(m_p \rho_N) = 3x10^{24} cm \approx 1000 kpc \tag{2.1}$$

Donde los rayos cósmicos puede estar algún tiempo en el halo galáctico más difuso, y esta es un limite menor a la distancia recorrida. En cualquier caso, $l \gg d \approx 0{,}1 kpc$, que es el grosor medio del disco de la galaxia. Esto implica que los rayos cósmicos se confinan en un proceso difusivo en el cual las partículas dan vueltas durante mucho tiempo antes de escapar al interior del espacio intergaláctico.

2.3. Espectro de energía

El espectro de varios elementos de los rayos cósmicos lo podemos ver en la figura 2.2. Las proporciones de las componentes mayores (con excepción del hierro) son relativamente constantes con la energía (Tabla 2.1). Estas son bien descritas por una inversa serie de potencias de la energía, con un flujo diferencial dado por:

$$\frac{dN}{dE} \propto E^{-(\gamma+1)} \tag{2.2}$$

El espectro es continuo por arriba de $E \sim 10^8 GeV$ con $\gamma \approx 1{,}7$, antes de esta energía el espectro va en aumento hasta $\gamma \sim 2{,}0$. Un punto importante a notar es el espectro del Boro, donde notamos que este es más empinado que el de sus parientes cercanos los núcleos de oxigeno y carbono. De hecho, todos los núcleos secundarios (es decir, aquellos que son producto de espalaciones) tienen un espectro significativamente mayor que los núcleos primarios. Las proporciones secundarias a primarias decrecen con le aumento de la energía. Esto nos dice que los rayos cósmicos de muy alta energía se difunden más rápido fuera de la galaxia.
Los rayos cósmicos están compuestos relativamente por protones en el rango de $10-100 GeV$ como se muestra en la tabla 2.1. La tabla nos muestra las fracciones de núcleos en relación con los protones en cuatro diferentes formas. Flujos que son normalmente usados, como en la columna (1): partículas por GeV por núcleos. Si definimos a la fracción de la columna (1) como F_A (es decir, $F_4 = 0{,}036$ para el núcleo de Helio), luego las fracciones de las otras columnas están relacionadas con la columna (1), así, por $2^\gamma F_A$ para la columna (2); AF_A para la columna (3) y $A^\gamma F_A$ para la columna (4).Estas relaciones dadas por una serie de potencias del espectro con $N_A(> N) \propto E^{-\gamma}$. Note que estas relaciones son para flujos integrales , es decir, para un número de partículas por encima del umbral de energía E. Las asignaciones M(medium), H(heavy) y VH(very heavy) son nomenclaturas de rayos cósmicos estándar para estos grupos de núcleos. Núcleos de luz (denotado L para $Z = 3$ o 5) son omitidos en la tabla ya que son extremadamente poco abundantes.

Cada una de las columnas son relevantes para determinada situación. La columna (1) (núcleos

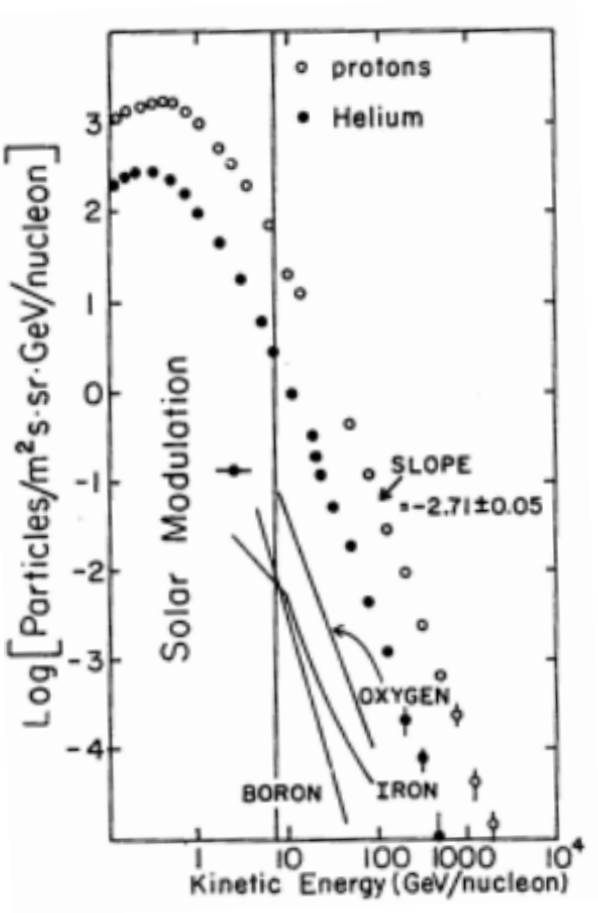

Figura 2.2: Espectro de energía de algunos componentes de los rayos cósmicos [8]

por energía por nucleón) es apropiado para cálculos de propagación ya que la energía por nucleón permanece esencialmente sin cambios en procesos de espalación. La rigidez $R \equiv pc/Ze$ es apropiada siempre que el radio de giro sea considerado relevante, como para la aceleración o para propagación a través de un campo geomagnético. De la columna (2) por ejemplo, seria esencial cuando, en una localización dada, para 1000 protones que atraviesan un campo geomagnético para alcanzar un detector en lo alto de la atmósfera, habrán cerca de 120 partículas alfa, 8 núcleos grupo M, 2 núcleos grupo H y un núcleo VH. La rigidez tiene las dimensiones de energía sobre carga. Unidades de GeV por carga son denotadas GV. El número de nucleones por GeV por nucleón (columna (3)) es la cantidad relevante para calcu-

Grupo de masa	$< A >$	(1) Partículas ($> E/A$)	(2) Partículas ($> R$)	(3) Núcleos ($> E/A$)	(4) Partículas ($> E/_{Ncls}$)
p	1	1	1	1	1
a	4	0.036	0.12	0.14	0.38
M ($Z = 6$ a 9)	14	0.0025	0.0083	0.0035	0.22
H ($Z = 10$ a 20)	24	0.0007	0.0023	0.017	0.15
VH ($Z = 21$ a 30)	56	0.0004	0.0013	0.022	0.40

Tabla 2.1: Fracciones de núcleos en relación con los protones [8]

lar flujos secundarios de partículas como los muones, antiprotones, piones, etc. ya que estas son esencialmente producidas por interacciones nucleón nucleón, incluso cuando los nucleones están ligados al núcleo (aunque pueden haber algunos efectos nucleares). Finalmente, la energía total por núcleos (columna (4)), es relevante para cascadas de aire ya que, el tamaño de la cascada refleja la energía total de la partícula incidente. Note que cuando los rayos cósmicos son clasificados por energía total por núcleos, menos de la mitad son protones.

2.4. Rayos cósmicos en la atmósfera

En la figura 2.3 se muestra los flujos verticales de los principales componentes de rayos cósmicos en la atmósfera en la región de energía donde las partículas son más numerosas (a excepción de los electrones, que son más numerosos cerca de su energía crítica, que es de unos $81MeV$ en el aire). Excepto protones y electrones. cerca de la parte superior de la atmósfera, todas las partículas se producen en interacciones de los rayos cósmicos primarios en el aire. Los muones y los neutrinos son productos de la cadena de descomposición de los mesones cargados, mientras que los electrones y los fotones se originan en las desintegraciones de los mesones neutros.

En la figura 2.3 podemos ver medidas de muones negativos. Como $\mu^+(\mu^-)$ son producidos en conjunto con $\nu_\mu(\overline{\nu}_\mu)$. La medición de muones cerca del máximo de la curva de intensidad para la protuberancia principal sirve para calibrar el haz atmosférico ν_μ. Debido a que los muones suelen perder casi 2 GeV al pasar a través de la atmósfera, la comparación cerca de la altitud de producción es importante para el rango sub-GeV de energías $\nu_\mu(\overline{\nu}_\mu)$.

El flujo de rayos cósmicos a través de la atmósfera se describe mediante un conjunto de ecuaciones en cascada acopladas con condiciones límite en la parte superior de la atmósfera para coincidir con el espectro primario. Se necesitan cálculos numéricos o de Monte Carlo para tener en cuenta con precisión los procesos de descomposición y pérdida de energía, y las dependencias energéticas de las secciones transversales y del índice espectral primario γ. Sin embargo, las soluciones analíticas aproximadas son útiles en regiones limitadas de energía. Por ejemplo, la intensidad vertical de los piones cargados de energía $E_\pi \ll \epsilon_\pi = 115GeV$ se calcula:

$$I_\pi(E_\pi, X) \approx \frac{Z_{N\pi}}{\lambda_N} I_N(E_\pi, 0) e^{-X/\Lambda \frac{XE_\pi}{\epsilon_\pi}} \tag{2.3}$$

donde Λ es la longitud característica de la atenuación exponencial del flujo de nucleones principal en la atmósfera. Esta expresión tiene un máximo en $X = \Lambda \approx 121 \pm 4g.cm^{-2}$ que corresponde a una altitud de 15 kilómetros. La cantidad $Z_{N\pi}$ es el momento ponderado por el espectro ponderado de la distribución inclusiva de los piones cargados en las interacciones de los nucleones con los núcleos de la atmósfera. La intensidad de piones de baja energía es mucho menor que la de los nucleones porque $Z_{N\pi} \approx 0{,}079$ es pequeño y porque la mayoría de los piones con energía mucho menor que la energía crítica ϵ_π se desintegran en lugar de interactuar.

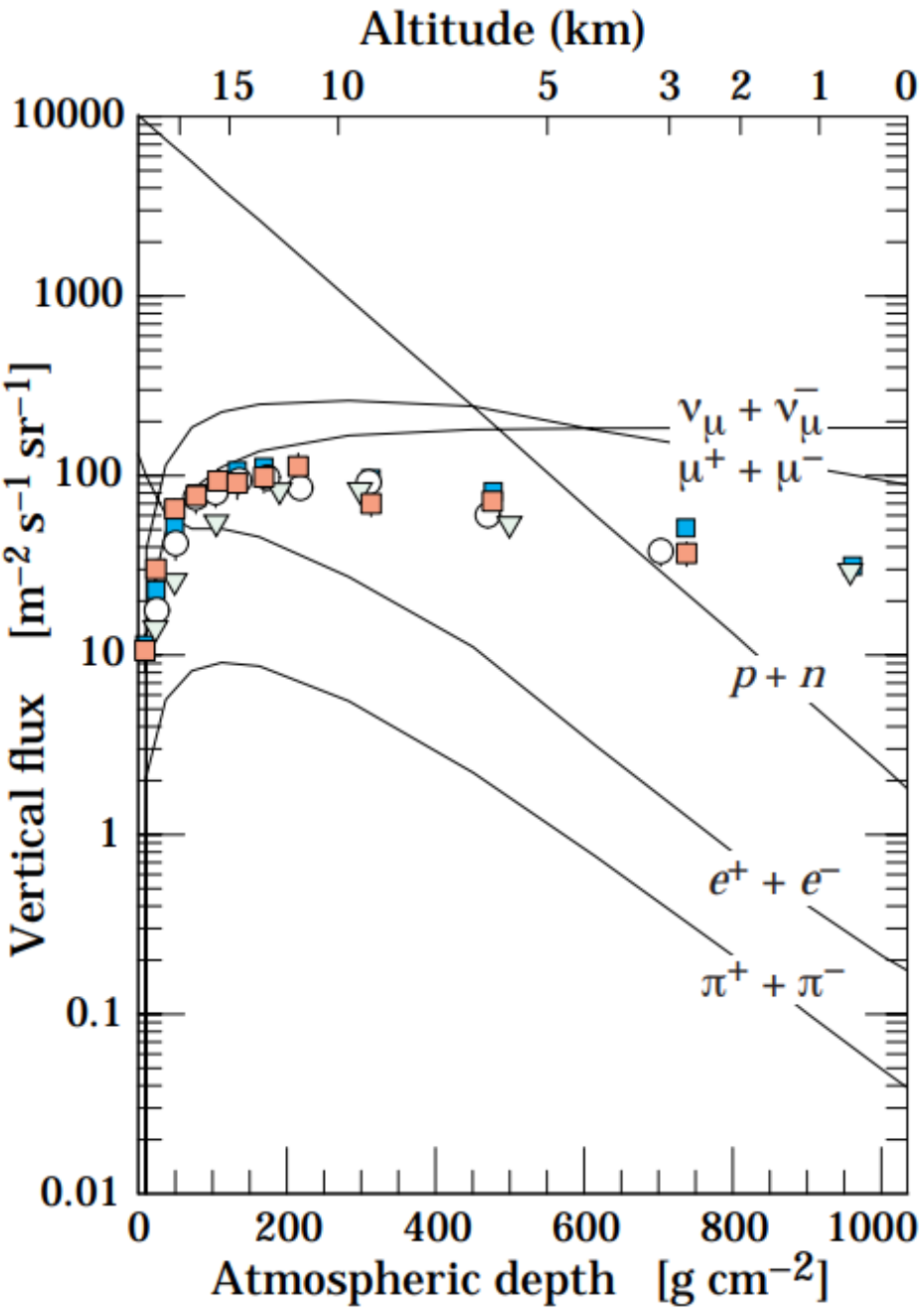

Figura 2.3: Flujos verticales de rayos cósmicos en la atmósfera con $E > 1GeV$ [9]

Capítulo 3

Física de Partículas

3.1. Decaimiento Beta

Para desarrollar una introducción eficiente y llegar a comprender la moderna física de partículas, se toma como ejemplo el muy usado decaimiento$-\beta$ cuyo propósito es apropiado para poder entender este tema. En el decaimiento$-\beta$ un núcleo con número de masa $A = Z + N$ con Z siendo el número de protones y N el número de neutrones sufre la siguiente transición:

$$A(Z, N) \rightarrow A'(Z + 1, N - 1) + e^{-} + \overline{\nu}_e \tag{3.4}$$

En un nivel más elemental, podemos entender esto como una transición de un nucleón único dentro del núcleo, así:

$$n \rightarrow p + e^{-} + \overline{\nu}_e \tag{3.5}$$

junto con un reordenamiento apropiado de los núcleos residuales. Protones y neutrones tienen la misma estructura, de modo que una descripción aún más fundamental del decaimiento$-\beta$ es posible. En el modelo de los quarks un proton consiste de tres quarks de valencia (2 quarks up (u) y un quark down (d)) junto con una gran cantidad indeterminada de gluones y pares de quark-antiquarks. Una lista de sabores de quarks y otras propiedades se muestran en la tabla 3.1. Los "sabores"de los quarks van en pares conocidos como generaciones, aunque el quark top aún no se ha descubierto experimentalmente. Ya que los quarks aparentemente no existen como particulas libres, sus masas son indefinidas. Podemos tomar apenas como la mitad de la masa de su correspondiente estado $q\overline{q}$ ("quarkonium"). es decir, el meson$-\phi$ para $s\overline{s}$ y el meson$-j/\psi$ para $c\overline{c}$. Los quarks también poseen un número bariónico de uno a tres, de modo que los bariones tienen la estructura de valencia quark qqq y los mesones $q\overline{q}$. En el

	Carga	Masa(GeV)
u (up)	+2/3	luz
d (down)	-1/3	luz
c (charm)	+2/3	1.6
s (strange)	-1/3	0.5
t (top)	+2/3	?
b (bottom)	-1/3	5

Tabla 3.1: Propiedades de los quarks y su nomenclatura [8]

nivel quark, además, el proceso del decaimiento$-\beta$ Figura 3.1 es realmente una transición elemental:

$$d \rightarrow u + e^{-} + \overline{\nu}_e \tag{3.6}$$

Lo que implica el cambio de estado del nucleón que contiene los quarks. Una opinión predominante es que, este es el nivel más fundamental en el cual una descripción de decaimiento$-\beta$ es posible. La carga fraccionada de los quarks aparentemente no puede existir como partículas libres, pero si como constituyentes de hadrones, entonces no es necesario postular otro nivel de estructura. Experimentalmente, los quarks no poseen estructura al menos para los niveles de $10^{-15}cm$, dos ordenes de magnitud más pequeños que el tamaño característico de los hadrones. En el modelo estándar estos son verdaderamente puntuales y básicamente en el mismo sentido que los leptones como electrones y muones.

De hecho, el diagrama de Feynman que describe el decaimiento$-\beta$ de un quark, es el mismo que el decaimiento$-\beta$ para el muón (Figura 3.2). Por supuesto, existen diferencias sustanciales en las tasas de decaimiento y distribución de las energías de las partículas secundarias, ya que los quarks están atados en el nucleón, solo que aquí hay diferencias entre decaimiento nuclear β y decaimiento de un neutrón libre debido a la unión nuclear. A pesar de eso, en casos particulares los hadrones contienen quarks pesados, tal como charm o bottom (aquí la masa del quark dominante es la masa del hadrón que lo contiene), incluso estas diferencias

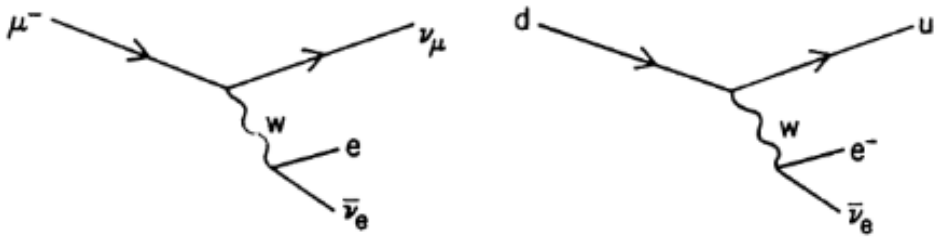

Figura 3.1: (a)Decaimiento del muón (b)decaimiento$-\beta$ de quark$-d$

son menores. El decaimiento del mesón charmed D^+, por ejemplo, puede ser entendido en el modelo de quark espectador, en la Figura 3.2. Aqui las formulas para la tasa de decaimiento y distribución de energía son esencialmente las mismas que para el decaimiento del muón siempre que la masa del muón sea remplazada por la masa del quark charmed, la cual es aproximadamente igual a la masa del mesón charmed. Los posibles productos del decaimiento leptonico son agrupados en pares llamados "isodobles debiles", que están relacionados con la generación de quarks (e para u y d y μ para c y s). Ya que los quarks vienen en tres colores, un decaimiento para $u\overline{d}$ es tres veces más probable que un decaimiento a uno de los canales leptónicos. Decaimientos de sabores pesados (es decir, para $\nu_\tau + \tau^+$) están prohibidos por la concertación de la energía ya que la masa de D es demasiado pequeña.

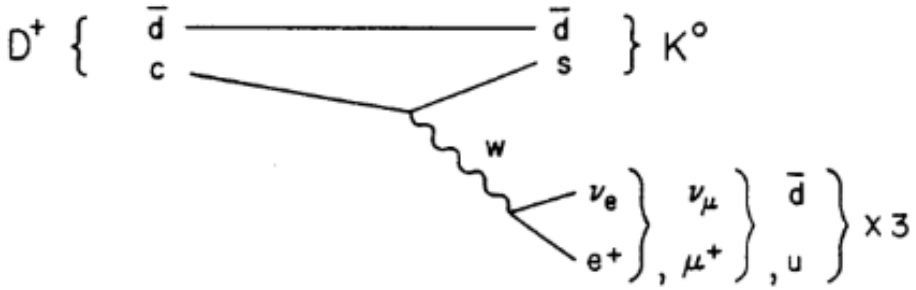

Figura 3.2: Decaimiento del mesón$-D^+$ cargado en el modelo de quark espectador (el $\overline{d}$ es el "espectador")

3.2. Cascadas atmosféricas

Cuando las partículas de alta energía interactúan con la materia, se crean nuevas partículas, que en general también poseen altas energías. Por lo tanto, se pueden formar cascadas o lluvias de partículas. Las denominadas cascadas atmosféricas extendidas o Extensive air shower (EAS por sus siglas en ingles) son las cascadas que se desarrollan en la atmósfera después de las interacciones de los rayos cósmicos. Es una cascada de partículas generada por la interacción de un rayo cósmico primario de muy alta energía, o por un fotón, con la parte superior de la atmósfera. El número de partículas al principio se multiplica, hasta alcanzar un valor máximo y posteriormente se atenúa como más y más partículas cayendo por debajo del umbral para una mayor producción de partículas. Los rayos cósmicos de alta energía sólo puede ser estudiado por la detección e interpretación de las EAS. Un rayo cósmico que induce una cascada atmosférica

Cuando los rayos cósmicos de ultra alta energía ingresan a la atmósfera e interactúan con los núcleos, las partículas secundarias que se producen en consecuencia, generan las EAS. Las cascadas secundarias se clasifican como cascada electromagnética, cuando son inducidas por partículas gammas o electrones de alta energía, y como cascada hadrónica si la partícula primaria es un núcleo o un hadrón, y esta a su vez posee tres componentes fundamentales, una componente electromagnética, una componente muonica y una componente hadronica[8].

La lluvia consiste de un núcleo de hadrones a muy alta energía que continuamente se alimenta de la parte electromagnética de la cascada, principalmente por fotones en decaimiento, de piones neutros y partículas eta. Cada uno de los fotones a muy alta energía generan una sublluvia electromagnética. Nucleones y otros hadrones de muy alta energía contribuyen en la producción de la componente hadronica. Piones cargados a baja energía y kaones en decaimiento alimentan la componente muonica Figura 3.3.

En cada interacción hadronica poco más de un tercio de la energía va hacia la componente electromagnética. Ya que la mayoría de los hadrones reinteractúan, la mayor parte de la energía primaria eventualmente encuentra su camino hacia la componente electromagnética. Además, debido a la rápida multiplicación de las cascadas electromagnéticas, electrones y

positrones son las partículas más numerosas en las EAS. Así la mayoría de la energía de las lluvias es eventualmente disipada por perdidas de ionización de electrones y positrones. Es entonces correcto pensar en la atmósfera como un calorímetro como ejemplo de un detector de estas lluvias atmosféricas.

A parte de la pequeña fracción $F(E_0)$ de energía perdida por neutrinos, la energía primaria, E_0 esta dada por la integral:

$$(1 - F) \times E_0 \sim \alpha \times \int_0^\infty dX N(X) \tag{3.7}$$

Donde $N(X)$ es el número de partículas cargadas en la cascada a una profundidad X (medida a lo largo del eje de la cascada) y α es la energía perdida por unidad de longitud en la atmósfera.

Aquí podemos tener en cuenta dos particularidades. La primera es que el número de muones de baja energía (1 a 10 GeV) incrementa con el desarrollo de la cascada hasta llegar a un valor constante ya que estos raramente interactúan catastróficamente, además, también pierden energía relativamente lento por ionización en el medio. La segunda particularidad es que el número de electrones y positrones decrece rápidamente después de alcanzar el valor máximo, esto debido a la radiación y producción de pares que subdividen la energía por debajo del valor de la energía critica, luego de que los electrones pierdan su energía restante por ionización rápidamente.

3.2.1. Cascada electromagnética

Las lluvias electromagnéticas contienen sólo electrones, positrones y partículas gamma, producido por Bremsstrahlung y procesos de producción de pares. Un modelo simplificado fue presentado por Heitler (en 1944) [10]. Después de una longitud de radiación d ($\simeq 37 g/cm^2$en aire), los positrones y electrones por encima de una energía crítica ($\simeq 85 MeV$) pierden la mitad de su energía emitiendo un fotón, mientras que los gammas producen otros electrones y positrones después de la misma longitud. En el desarrollo máximo de la cascada (X_{max}) el número total de partículas generadas es proporcional a la energía del rayo cósmico primario.

Cuando la energía de las partículas individuales se vuelve más pequeña que la energía crítica, las pérdidas por ionización se vuelven predominantes y los electrones (y positrones) son absorbidos en la atmósfera.
Usando un modelo muy simple del desarrollo de tales cascadas (energía incidente E_0, electrones con energías E_e por encima de la energía crítica E_c emitiendo fotones después de una longitud de radiación X_0, fotones con energías $E_\gamma > E_c$ creando pares de electrones-positrones después de $1X_0$, la energía se comparte por igual entre las partículas en cada paso), se pueden determinar aproximadamente algunas propiedades importantes:

- El número N de partículas en un paso de la cascada así como su energía E depende de la posición x en el material: $N \approx 2^{x/X_0} \approx E_0/E$
- la creación de nuevas partículas se detiene cuando todas las partículas cargadas caen por debajo de la energía crítica E_c. Por lo tanto, el número total de partículas creadas en la cascada es $N_{tot} \approx 2E_0/E_c$ y la posición de la cascada máxima en unidades de X_0 se puede aproximar con $x \approx ln(E_0/E_c)/ln2$
- Para absorber el $95\,\%$ de la cascada, el material debe tener un espesor de aproximadamente $15 - 20$ veces X_0.

Como se menciono esta es una aproximación de como se componen las cascadas electromagnéticas la descripción realista del desarrollo de estas cascadas es mucho más complicada.

3.2.2. Cascada hadronica

Se puede aplicar un modelo simplificado similar a las duchas de aire hadrónicas[11]. La probabilidad de interacción del rayo cósmico primario depende de su sección transversal inelástica en el aire, que, a su vez, es función de la energía primaria. Debido a los varios canales de interacción y secciones transversales posibles, incluso para el mismo tipo de rayo cósmico de una energía específica la altura de interacción no es fija. Por lo tanto, surgen fluctuaciones en las cascadas.

las partículas secundarias en las cascadas hadrónicas se agrupan en tres componentes principales:

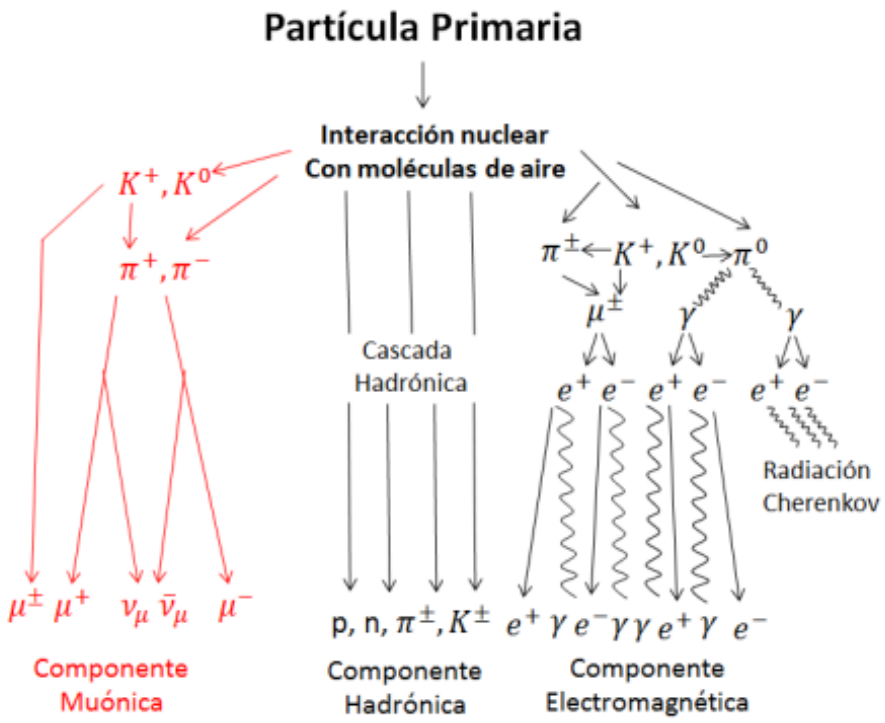

Figura 3.3: Representación esquemática de una cascada de partículas

El componente hadrónico

Es el que transporta una gran fracción de la energía de la cascada. Debido a los altos momentos que posee , los hadrones en la cascada (π^+, π^-, π^0, kaones pero también protones y neutrones) se ensamblan alrededor del eje de la lluvia.

El componente muonico

Los piones cargados por encima de $100GeV$ pueden interactuar o decaer (vida media $\sim 2{,}6 * 10^{-8}s$) generando el componente muónico. Los muones en las lluvias de aire son ge-

nerados no solo por piones cargados, sino también por descomposición de kaones. Los más energéticos ($> 100GeV$) se originan en la primera etapa del desarrollo de la cascada y penetran la atmósfera completa casi sin perder su energía por ionización o Bremsstrahlung. Por el contrario, los muones de baja energía se descomponen en electrones (o positrones) y neutrinos.

El componente electromagnético

La lluvia es generada principalmente por piones neutros, que se desintegran casi instantáneamente en dos fotones, induciendo así subduchas electromagnéticas. Además, las cascadas hadrónicas tienen una fracción electromagnética, que se puede aproximar como [12]:

$$f_{em} = 1 - (\frac{E}{E_0})^{k*1} \tag{3.8}$$

donde E es la energía inicial del hadrón primario, E_0 es la energía mínima necesaria para interacciones inelásticas y varía entre $0,7GeV$ (para hierro) y $1,3GeV$ (para plomo) y el parámetro k varía entre $0{,}80$ y $0{,}87$, dependiendo de la multiplicidad media de piones neutros y la pérdida de energía debido a los piones neutros en una sola interacción. Ambos parámetros deben obtenerse experimentalmente para una configuración de calorímetro dada.

Capítulo 4

Muones

Es una partícula de carga negativa descubierta por Carl Anderson en 1936, es una partícula elemental de la familia de los leptones, que al igual que el electrón, que tiene una carga negativa y espín $1/2$; cuenta con una masa de $105,7 MeV/c^2$, aproximadamente 200 veces más masivo que el electrón, creando un gran interés en el estudio de estos, ya que al ser más masivos que los electrones sus trayectorias se ven menos afectadas por el campo magnético terrestre y por tanto es más difícil que se desvíen de su trayectoria original. Al muón y a su antipartícula se los denota como μ^- y μ^+, respectivamente. Los muones son casi estables y poseen una pequeña sección transversal para las interacciones, por lo que son muy penetrantes. Por ello, los muones son conocidos por ser llamados el "componente penetrante" de los rayos cósmicos. Además, ya que están cargados, estos son relativamente fáciles de detectar, de este modo, los muones nos proporcionan información importante tanto de la atmósfera y también bajo tierra.

4.1. Muones en las cascadas atmosféricas extendidas

Como vimos en el capitulo anterior, la componente muonica consiste en muones de alta energía provenientes del decaimiento de piones y kaones de la cascada de hadrones que forman la tercera componente de la lluvia Figura 3.3.

Las primeras interacciones hadrónicas dan origen a piones cargados y neutros los cuales, en los altos niveles de la atmósfera, tiene una mayor probabilidad de decaer que de interactuar con los escasos átomos presentes. Los piones neutros alimentan la componente EM (electromagnética) al producir fotones y electrones muy energéticos según las siguientes reacciones:

$$\pi^0 \rightarrow \gamma\gamma[Probabilidad : 98{,}8\,\%] \tag{4.9}$$

$$\pi^0 \rightarrow \gamma e^+ e^-[Probabilidad : 1{,}2\,\%] \tag{4.10}$$

Posteriormente, los piones cargados decaen en muones energéticos que dan origen a la componente muónica de la lluvia, la cual es mucho mayor en estas lluvias respecto a las iniciadas por partículas EM:

$$\pi^+ \rightarrow \mu^+ \nu_\mu[Probabilidad : 99{,}99\,\%] \tag{4.11}$$

$$\pi^+ \rightarrow e^+ \nu_e[Probabilidad : 0{,}01\,\%] \tag{4.12}$$

Los mesones extraños, principalmente kaones, son también fuente de muones luego de su decaimiento:

$$K^+ \rightarrow \mu^+ \nu_\mu[Probabilidad : 63{,}43\,\%] \tag{4.13}$$

$$K^+ \rightarrow \pi^+ \pi^0[Probabilidad : 21{,}13\,\%] \tag{4.14}$$

$$K^+ \rightarrow \pi^+ \pi^+ \pi^-[Probabilidad : 5{,}6\,\%] \tag{4.15}$$

$$K^+ \rightarrow \pi^0 e^+ \nu_e[Probabilidad : 4{,}9\,\%] \tag{4.16}$$

Por otro lado, mesones extraños, principalmente kaónes, se generan también en la lluvia hadrónica y su decaimiento junto con sus conjugados de carga, que a su vez decaen en más muones, salvo el último que desencadena una sublluvia EM. Y finalmente, mesones con una vida media mucho más corta, decaen antes de interactuar produciendo kaones y muones de alta energía.

El efecto temporal relativista, junto a la comparativamente larga vida de los muones ($\tau = 2,197 \times 10^{-6}$s), sumado a la facilidad de detección de los mismos, hacen que el estudio de la cascada muónica sea de mucho interés al contener información relevante de lo ocurrido en las primeras interacciones hadrónicas, y reflejan en forma más directa las propiedades del hadrón inicial. Para los muones producidos en las primeras interacciones, su alta energía hace que sean importantes los efectos radiactivos, caracterizados por pequeñas secciones eficaces

y grandes fluctuaciones en la energía de las partículas resultantes. las pérdidas de energías por la interacción con la atmósfera no serán del todo despreciables, como lo es para aquellos muones con energías por debajo de la escala de TeV. Sin embargo, el poder de frenado de los muones es extremadamente bajo respecto a las energías típicas $\approx 2MeVcm^2g^{-1}$ y para muones con energías del orden de decenas de TeV el poder de frenado crece hasta sólo $\approx 30MeVcm^2g^{-1}$ indicando que el muón tiene un alto poder de penetración de la materia perdiendo un porcentaje muy bajo de su energía inicial [13].

4.2. Espectro de energía de los muones

Los muones son las partículas cargadas más numerosas al nivel del mar (ver Fig. 2.3). La mayoría de los muones se producen en lo alto de la atmósfera (normalmente a $15km$) y pierden unos $2GeV$ por ionización antes de llegar al suelo. Su energía y su distribución angular reflejan una convolución del espectro de producción, pérdida de energía en la atmósfera y descomposición. Por ejemplo, los muones de $2,4GeV$ tienen una longitud de decaimiento de $15km$, que se reduce a $8,7km$ por la pérdida de energía. La energía media de los muones en el suelo es $\approx 4GeV$. El espectro de energía es casi plano por debajo de $1GeV$, aumenta gradualmente para reflejar el espectro primario en el rango de 10 a $100GeV$ y aumenta aún más a energías más altas porque los piones con $E_\pi > \epsilon_\pi$ tienden a interactuar en la atmósfera antes de desintegrarse. Asintóticamente ($E_\mu \gg 1TeV$), el espectro de energía de los muones atmosféricos es una potencia más pronunciada que el espectro primario. La integral de la intensidad de los muones verticales por encima de $1GeV/c$ a nivel del mar es $\approx 70m^{-1}s^{-1}sr^{-1}$, con mediciones recientes, favoreciendo una menor normalización en un $10-15\,\%$. Los experimentadores están familiarizados con este número en la forma $I \approx 1cm^{-2}min^{-1}$ para detectores horizontales. La distribución angular general de los muones en el suelo en función del ángulo cenital θ es $\propto cos^2\theta$, que es característico de los muones con $E_\mu \sim 3GeV$. A menor energía, la distribución angular se vuelve cada vez más pronunciada, mientras que a mayor energía se aplana, acercándose a una distribución $sec\theta$ para $E_\pi \gg \epsilon_\pi$ y $\theta < 70^o$.

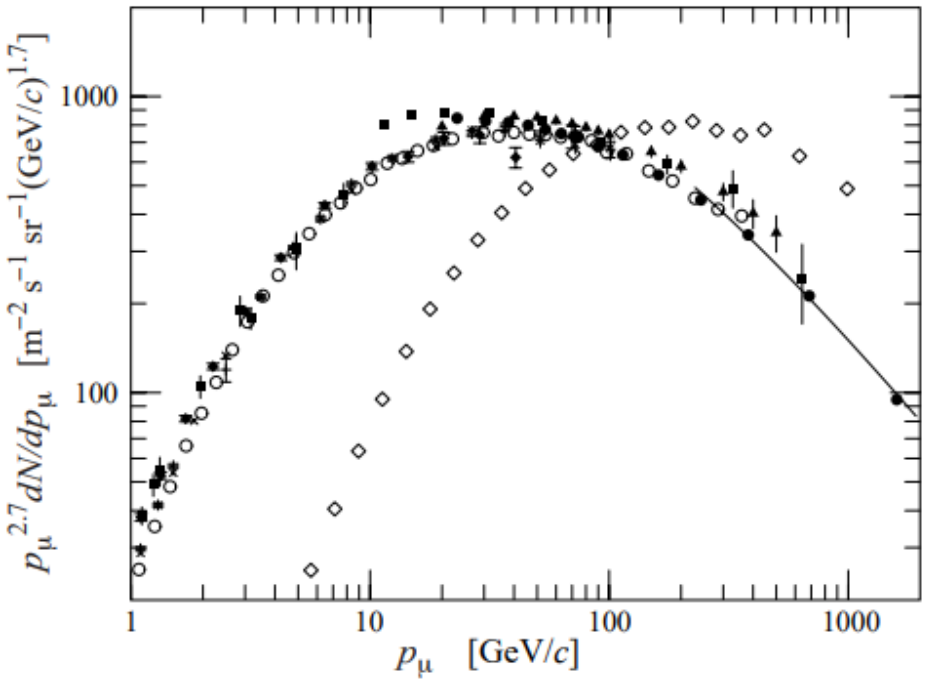

Figura 4.1: Espectro de muones en $\theta = 0^o$ [8]

La Figura 4.1 muestra el espectro de energía de los muones al nivel del mar para dos ángulos. En ángulos grandes, los muones de baja energía se desintegran antes de alcanzar la superficie y los piones de alta energía se desintegran antes de interactuar, por lo tanto, la energía media de los muones aumenta. Una fórmula de extrapolación aproximada válida cuando la descomposición del muón es insignificante ($E_\mu > 100/cos\theta GeV$) y la curvatura de la Tierra puede despreciarse ($\theta < 70^o$) es:

$$\frac{dN_\mu}{dE_\mu d\Omega} \approx \frac{0{,}14 E_\mu^{-2{,}7}}{cm^2.s.sr.GeV} * \left(\frac{1}{1 + \frac{1{,}1 E_\mu cos\theta}{115 GeV}} + \frac{0{,}054}{1 + \frac{1{,}1 E_\mu cos\theta}{850 GeV}}\right) \tag{4.17}$$

donde los dos términos dan la contribución de piones y kaones cargados.

4.3. Interacción muón-materia

Las partículas cargadas y la radiación en general presentan interacciones a medida que atraviesan la materia, en materiales solidos y en la atmósfera. En general, hay dos principales rasgos que caracterizan este fenómeno, estos son: (1) la perdida de energía que tiene la partícula y (2) la deflexión de la partícula de su dirección incidente. Lo anterior, ocurre como resultado principal de las colisiones inelásticas con los electrones atómicos del material y de la dispersión elástica de los núcleos [14].

Cualquier partícula cargada interactúa con la materia a través de diferentes procesos, como radiación, producción de pares, bremsstrahlung y efectos fotonucleares. En el caso de los muones, esta interacción está liderada por los procesos de radiación y la pérdida de energía media, con respecto a la masa por unidad de área (x) del material en unidades de $MeV.g^{-1}.cm^2$, esto se define como la capacidad de penetración de un muón, que usa una cantidad denominada poder medio de frenado, que describe como disminuye la energıa de un muón conforme atraviesa la materia. El poder de frenado se puede escribir como [13]:

$$\left\langle \frac{-dE}{dx} \right\rangle = a(E) + b(E)E \tag{4.18}$$

Donde $b(E)$ corresponde a la suma de la producción de pares, bremsstrahlung y contribuciones fotonucleares, además, su efecto incrementa con la energía. En la figura 4.3 se presentan los valores de ajuste para este parámetro en función de la energía y para un muón interactuando con roca estándar. Y $a(E)$ representa las pérdidas por ionización y viene dada por la fórmula de Bethe-Bloch:

$$\left\langle \frac{-dE}{dx} \right\rangle = K\frac{Z}{A}\frac{1}{\beta^2}\left[\frac{1}{2}ln\frac{2m_e c^2\beta^2\gamma^2 Q_{max}}{I^2} - \beta^2 - \frac{\delta}{2} + \frac{1}{8}\frac{Q_{max}^2}{(\gamma Mc^2)^2}\right] + \triangle\left|\frac{dE}{dx}\right| \tag{4.19}$$

donde el ultimo término se refiere a correcciones de alta energía por Bremmsstrahlung y está dado por:

$$\Delta\left|\frac{dE}{dx}\right| = \frac{K}{4\pi}\frac{Z}{A}\alpha\left[ln\frac{2E}{M_\mu c^2} - \frac{1}{3}ln\frac{2Q_{max}}{m_e c^2}\right]ln^2\left(\frac{2Q_{max}}{m_e c^2}\right) \tag{4.20}$$

cuyos valores pueden ser estimados a partir de gráficos como el presentado en la figura 4.2 (potencia de frenado del muón en cobre).

Donde Z el número atómico, A es el número másico, K es una constante con un valor de $K \approx 0{,}307\frac{Mevcm^2}{mol}$, Q_{max} es la máxima energía recibida por un electrón en una colisión, I es la energía media de excitación, δ es una corrección por perdidas de energía de ionización, β es la velocidad de la partícula en relación con la de la luz c y γ es el factor de Lorentz.

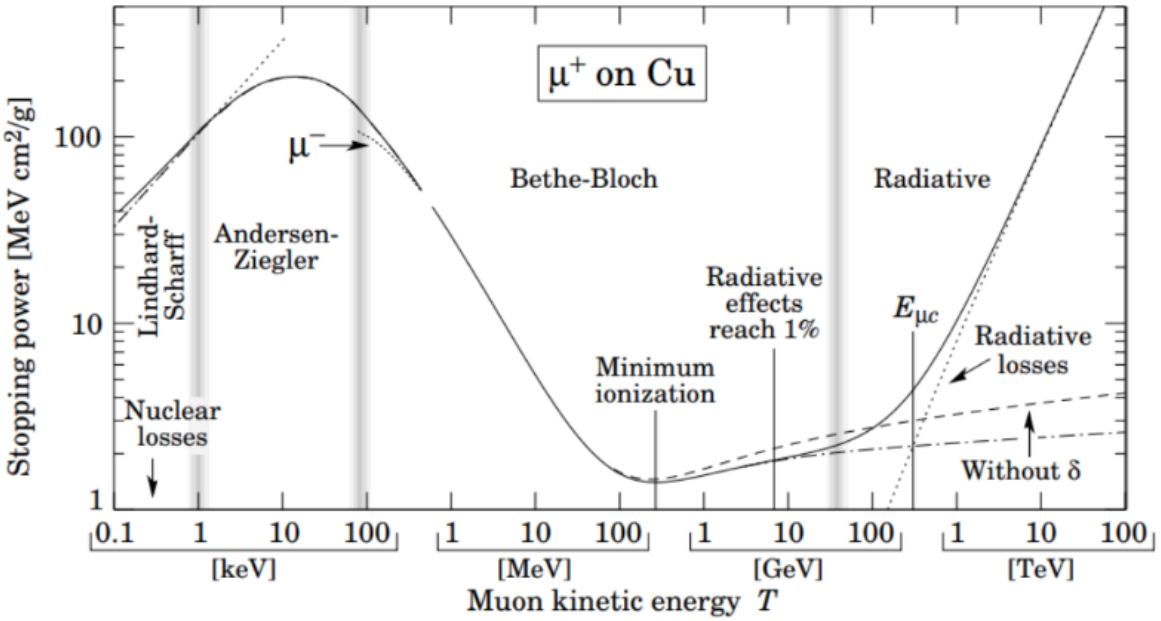

Figura 4.2: Gráfico de valores de potencia de frenado en función de le energıa, para un muón penetrando en cobre. [13]

La sección transversal diferencial de un muón que experimenta por radiación de frenado o Bremsstrahlung viene dada por [13]:

$$\left(\frac{d\sigma}{d\nu}\right)_{Bremsstrahlung} = \alpha\left(2Z\frac{m_e}{M_\mu}r_e\right)^2\left(\frac{4}{3} - \frac{4\nu}{3} + \nu^2\right)\frac{\Phi(\delta)}{\nu} \tag{4.21}$$

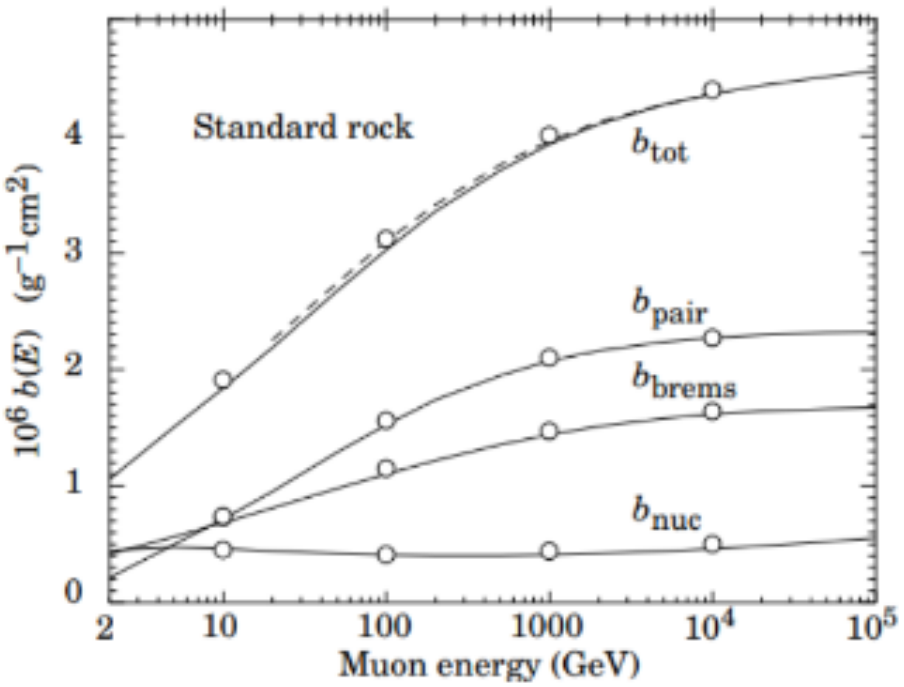

Figura 4.3: Valores de ajuste del termino $b(E)$ asociado a procesos radiativos para muones en roca estándar (densidad $\rho = 2,6g/cm^3$). $b_{tot} = b_{par} + b_{brems} + b_{nuc}$.[13]

donde ν es la fracción de energía del muón que se transfiere al fotón y $\Phi(\delta)$ esta definido como:

$$\Phi(\delta) = ln\left(\frac{BM_\mu Z^{-1/3}/m_e}{1+\delta\sqrt{e}BZ^{-1/3}/m_e}\right) - \Delta_n(\delta) \tag{4.22}$$

donde B es una constante que toma el valor de 182,7, $\delta = M_\mu^2\nu/2E(1-\nu)$ y la corrección por apantallamiento nuclear esta dada por:

$$\delta_n = ln\left(\frac{D_n}{1+\delta(D_n\sqrt{e}-2)/M_\mu}\right) \tag{4.23}$$

4.4. Muongrafía

La alta capacidad de penetración de los muones los convierte en un novedoso método físico para la captura de imágenes remotas por muongrafía, el uso de este método se basa en partículas de alta energía para la obtención de imágenes que son producidas por rayos cósmicos en lluvias de partículas en la atmósfera terrestre. Partículas de muones que se detectan después de pasar por los medios de interés. Si los objetos estudiados son sólidos, sus tamaños pueden variar desde metros hasta kilómetros. En términos de capacidad de penetración, la muografía se puede ubicar entre los métodos basados en rayos X y los que usan ondas sísmicas. Los objetos más famosos mapeados con muongrafía son pirámides (por ejemplo, la pirámide de Khufu en Giza en Egipto [15]) y volcanes (por ejemplo, el monte Etna en Italia [16]). Una clara ventaja de la muografía en comparación con los métodos sísmicos es que los muones, a diferencia de las ondas sísmicas, no se reflejan en las interfaces geológicas. Además, el fenómeno de dispersión es un problema menor y necesita consideración solo en muones de baja energía. Los datos con incertidumbre deben corregirse utilizando la topografía. Las variaciones de densidad más bajas observables para muongrafía con un nivel significativo de 3σ (un nivel de significación típico en física) son alrededor del $2\,\%$ a 150m, $4\,\%$ a 300m, y $10\,\%$ a 700m de profundidad, respectivamente. Si estos números se extrapolan a profundidades por debajo de los 100m, es probable que las diferencias de densidad media en el rango del $1\,\%$ estén dentro de la capacidad de observación de muongrafía. También vale la pena señalar que la diferencia del $1\,\%$ en la densidad media de la roca da como resultado una diferencia de aproximadamente el $3\,\%$ en el flujo de muones. Esto indica que las mediciones del flujo de muones son muy sensibles a las variaciones de densidad de las rocas [17]. El reto en esta técnica, es poder medir con alta precisión el ángulo de dispersión asociado ya que el nivel de deflexión estará relacionado con el número atómico del material (Z), este ángulo tiene una distribución Gaussiana, con valor medio 0 y RMS (proyectado en un plano) dependiendo de:

$$\sigma_\theta = \frac{13{,}6MeV}{\beta cp(MeV/c)}\sqrt{\frac{L}{X_0}}\left(1 + 0{,}0038ln(\frac{L}{X_0})\right) \tag{4.24}$$

En donde vemos que, p es el momento del muón, L es el grosor del material con el que

interaccionan los muones, β y c son constantes y X_0 es la longitud de radiación que es distancia bajo la cual electrones a muy alta energía pierden un factor de $1/e$ de su energía por bremsstrahlung (esta radiación depende del número atómico del material Z):

$$X_0 = \frac{716{,}4g.cm^{-}3}{Z(Z+1)ln(287/\sqrt{Z})} \tag{4.25}$$

generalmente, las partículas cargadas sufren deflexiones de sus trayectorias debido a interacciones de tipo Coulomb, por ello a este proceso también se lo conoce como muongrafía por dispersión o scattering. Además, como el ángulo de dispersión depende directamente del número atómico del material, esta técnica solo se puede aplicar a objetos altamente densos y un número atómico (Z) apreciable.

Para la contabilización de la naturaleza aleatoria de dispersión, las trayectorias de los muones se desvían un ángulo $\delta\theta$ desde su dirección original θ, con una probabilidad dada por una distribución de Rayleigh [18]:

$$P(\delta\theta) = \frac{\delta\theta}{\sigma_\theta^2} e^{-\frac{\delta\theta^2}{2\sigma_\theta^2}} \tag{4.26}$$

La ecuación (4.24) nos muestra que la dispersión, σ_θ, de un muón depende enormemente de su energía; en consecuencia, no puede ser directamente aplicado a largas trayectorias a través de la roca, ya que los muones pierden energía conforme va avanzando. Para tal situación, cuando un muón con $E > E_{min}$ penetra en la roca, se dispersa cada vez más mientras su energía disminuye [19]. Entonces es necesario integrar una forma diferencial para esta ecuación. Sin embargo, debido a la dependencia E^{-1} de σ_θ, la mayor parte de la dispersión ocurre en la última parte del trayectoria donde la energía del muón es mínima. Como se muestra en la figura 4.4, es necesaria una energía mínima de $200GeV$ para que un muón atraviese una capa de roca de $300m$ y, a este nivel de energía, la dispersión, σ_θ, es de $0{,}53mrad$, después de un recorrido de 10 metros de largo. La dispersión neta se puede obtener sumando el cuadrado de $\sigma'_\theta s$ (es decir, las varianzas), calculadas a lo largo de la trayectoria del muón y teniendo en cuenta la pérdida de energía. Se puede apreciar como los muones con energías superiores a 10^3GeV, pueden llegar a atravesar varios kilómetros de roca antes de perder toda su energıa.

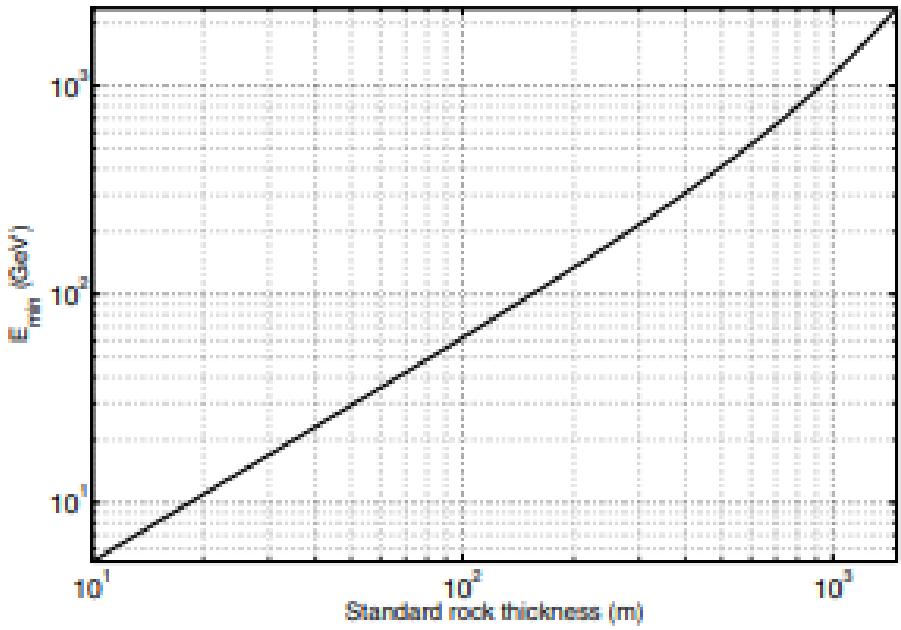

Figura 4.4: Energía mínima E_{min} necesaria para que un muón logre atravesar una cierta cantidad de roca estándar

4.5. Técnicas de detección

Con el paso de los años, las técnicas experimentales de detección han ido evolucionando y adaptándose para el análisis de los distintos valores de energía que varían dependiendo del tipo de radiación a estudiar. En física de partículas los diversos métodos implementados y la gran cantidad de mecanismos existentes para ello hoy en día nos proporcionan una mayor viabilidad a la hora de detectar este tipo de partículas producto de rayos cósmicos. Actualmente existen dos formas de detección en el estudio de los rayos cósmicos, estos son: métodos de detección directa y de detección indirecta.

4.5.1. Detección Directa

Este método se basa en aprovechar el flujo de radiación incidente, ya que para energías que estén por debajo del umbral de los $10^{15}eV$ es posible obtener un flujo lo suficientemente alto

como para ser detectado utilizando uno de los métodos de detección directa. Los muones en este caso, al ser partículas que poseen carga, pueden sufrir interacciones por ionización con la materia, razón por la cual al penetrar la materia los muones atraen, por interacción Coulombiana, a los electrones cercanos, produciendo ionizaciones en el material y perdiendo así una fracción de su energía en su recorrido debido principalmente a este proceso. Este tipo de procesos se pueden detectar con el uso de tubos Geiger Muller o centelladores, instrumentos muy utilizados para la medición de radiación ionizante. Otra técnica comúnmente usada en la practica es el uso de la técnica encargada de recolectar luz coherente, es decir, cuando una partícula energética viaja más rápido que la luz en un medio, esta emite una onda de choque, que es llamada luz Cherenkov. Un ejemplo de un medio común puede ser el agua que es económica y transparente, características adecuadas que permiten la detección de la luz e indirectamente la detección de partículas de altas energías[20].

Detectores de Centelleo

Los detectores de centelleo poseen la propiedad de absorber energía (a través de interacciones de Coulomb) y vuelven a emitir esa energía en forma de radiación electromagnética (luz de centelleo) cuando por él pasa radiación ionizante (como por ejemplo muones). Un ejemplo de este material son los plásticos centelladores los cuales son un polímero que se excita y se des-excita con el paso de las partículas cargadas. En general, este tipo de material hace que los detectores sean adecuados para cualquier geometría siendo una excelente opción en cuanto a diseño físico. Independientemente del montaje utilizado, los fotones son emitidos por el material centellador cuando los muones lo atraviesan, esta luz emitida debe ser transformada en señal eléctrica para a su vez poder ser recolectada (para realizar un conteo) por ejemplo por un fotomultiplicador o un fotodiodo, este puede ser fotomultiplicador de Silicio (SiPMs) o tubo fotomultiplicador (PMT), este se encarga de absorber la luz emitida por el centellador y los reemite como electrones por efecto fotoeléctrico, esta señal eléctrica es en un principio digitalizada y posteriormente analizada. A estos dos componentes en conjunto (material centellador y detector de luz) es a lo que llamamos detector de centello.

el uso de centelladores es el método de detección más convencional en muografía, principalmente debido a su estabilidad, además no tiene grandes requerimientos previos para su uso, a diferencia de las emulsiones nucleares o detectores gaseosos, cuya estabilidad es más dificil de garantizar, con centelladores plásticos se puede diseñar fácilmente sistemas de detección de muones que sean de gran tamaño o de geometrías complejas y también ser adecuados para ambientes hostiles ya que no son sensibles a las condiciones ambientales y adicionalmente a eso, son un material de bajo costo con alta eficiencia de detección que permite ser uno de los materiales favoritos a la hora de trabajar con este tipo de experimentos[21].

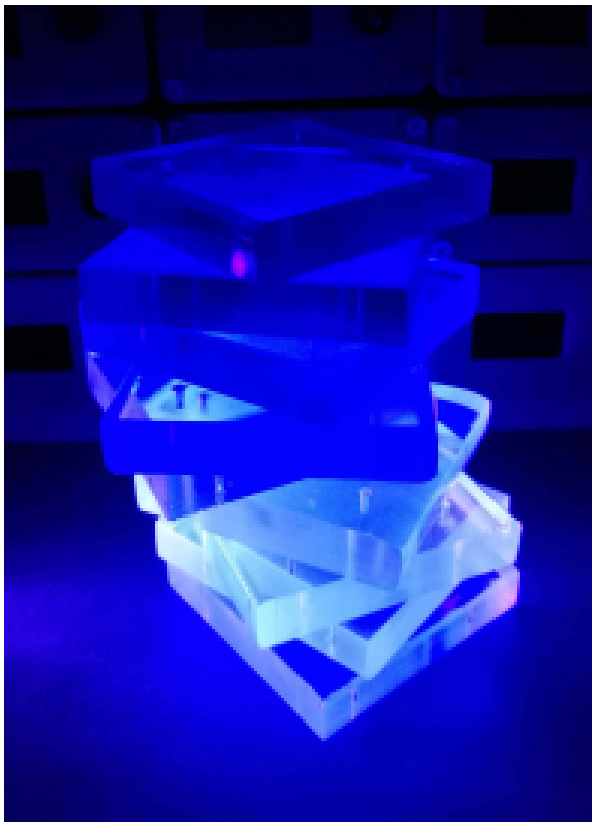

Figura 4.5: Linterna ultravioleta iluminando material centellador[21]

El funcionamiento de un detector de tipo centelleo se dispone de la siguiente manera: En un principio la radiación ionizante (como los muones) incide sobre el material centellador y le

transfiere la energía que estos pierden en su trayecto, parte de esa energía perdida la absorbe el material y la libera en forma de fotones a través de centelleo, con ayuda de reflectores, guías de luz o fibras ópticas (WLS por sus siglas en ingles) se lleva la luz producida hacia el fotomultiplicador para realizar el respectivo conteo de fotones, es el fotocátodo del mismo quien absorbe los fotones y emite electrones para convertirlos en señal eléctrica y digitalizarla para su posterior análisis. Podemos ver un esquema del proceso en la figura 4.6.

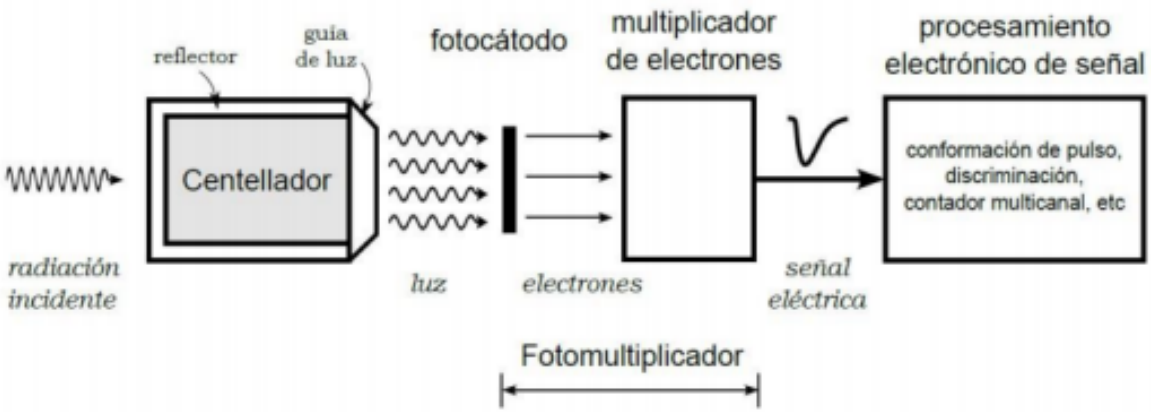

Figura 4.6: Detector de centelleo acoplado a fotomultiplicador[22]

Detectores Cherenkov

Estos detectores aprovechan la conocida radiación de Cherenkov, la cual es radiación electromagnética (también llamada luz Cherenkov) que se emite cuando una partícula cargada pasa a través de un medio dieléctrico a una velocidad mayor que la velocidad de la luz en ese medio. El procedimiento básico de reconstrucción de eventos es análogo al que se usa para detectores de partículas, pero los datos Cherenkov en el aire contienen más información sobre la lluvia longitudinal. Debido a la buena transparencia general de la atmósfera para la parte óptica de la emisión de Cherenkov (poca absorción y dispersión), la luz recolectada por el detector en tierra contiene fotones de todas las etapas de la lluvia, junto con toda la trayectoria. Por lo tanto, el componente Cherenkov lleva la historia de una lluvia completa al observador a nivel del suelo, revelando así una imagen tridimensional de un evento.

Por lo tanto, las mediciones de Cherenkov representan de hecho una especie de calorimetría Cherenkov atmósferica de la lluvia donde en principio todas las etapas de desarrollo son accesibles a través de los fotones.
Una ventaja importante del método de detección de los detectores Cherenkov atmosféricos sobre la detección de partículas, es que debido a la cantidad mucho mayor de fotones de una ráfaga de luz Cherenkov de una ducha que llega al detector en comparación con el número de partículas que golpean un detector de partículas convencional, es efectivamente, que el estallido de fotones no está sujeto a las fluctuaciones poissonianas como lo es el conteo de partículas. Las fluctuaciones de los detectores Cherenkov son, por lo tanto, de una naturaleza diferente. Son causados por las fluctuaciones de los procesos físicos reales que tienen lugar en una lluvia y el flujo de fotones de fondo superpuesto que fluctúa fuertemente. El conteo neto de fotones de la lluvia se estima luego restando el conteo de fondo de la señal medida.

Fotomultiplicadores

Los fotomultiplicadores (PMT por sus siglas en ingles), son dispositivos de tubos de electrones que convierten luz en corriente eléctrica apreciable. Son extremadamente sensibles y en física nuclear y de altas energía, están muy relacionados con detectores de centelleo, a pesar de que sus usos son bastante variados. Se pueden obtener pulsos eléctricos a partir de unos pocos cientos de fotones visibles. Es sin embargo en este contexto que se discute el diseño básico y propiedades de los fotomultiplicadores[23].

En la figura 4.7 se muestra se muestra un diagrama esquemático de un fotomultiplicador típico. Consiste de un cátodo de material fotosensible seguido por un sistema de colección de electrones, un sector de multiplicación de electrones (llamado usualmente dinodo) y finalmente un ánodo del cual se puede extraer la señal final. Todas las partes están usualmente contenidas en un tubo de vidrio para que todo el fotomultiplicador tenga la apariencia de un antiguo tuvo de electrones.
Durante la operación un alto voltaje es aplicado al cátodo, dínodo y ánodo tal que un potencial escalera es colocado a lo largo de la longitud de la estructura que componen cátodo-

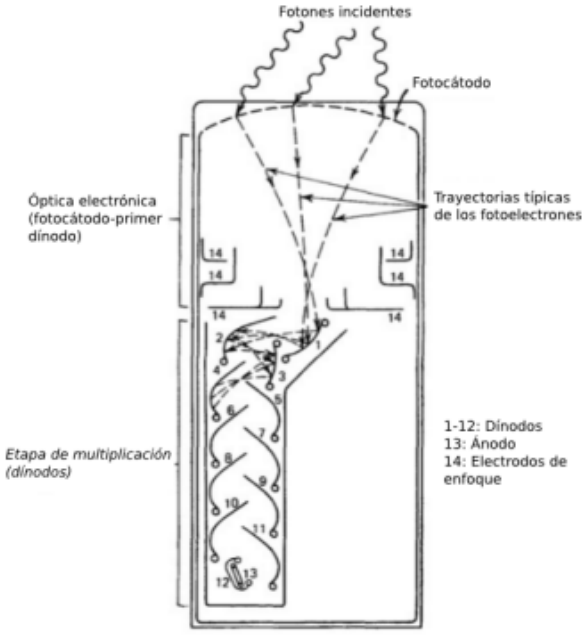

Figura 4.7: Esquema tipico de un PMT[23]

dínodo-ánodo. Cuando un fotón incide sobre el fotocátodo, un electrón es emitido vía efecto fotoelectrico, es debido al voltaje aplicado, el electrón es luego dirigido y acelerado hacia el primer dínodo, donde al golpearlo, se transfiere parte de su energía a los electrones en el dínodo. Esto causa electrones secundarios al ser emitidos que a su vez son acelerados hacia el siguiente dínodo donde más electrones son lanzados y son mucho más acelerados. Por lo tanto, se crea una cascada de electrones a lo largo de la cadena de dínodos, en el ánodo esta cascada es recolectada y se obtiene una corriente la cual sera amplificada y posteriormente analizada. Los PMT pueden ser operados en modo continuo, es decir, bajo iluminación constante, o en modo de pulsos como es el caso del centelleo. Por otro lado, si los sistemas cátodo y dínodo se asumen que son lineales, la corriente de salida del fotomultiplicador sera directamente proporcional al número de fotones incidentes. La radiación del detector se produce por acoplar un centellador a un PMT (el centellador produce fotones en proporción a la energía depositada en el centellador) sería así capaz de proporcionar no solamente información de la presencia de partículas sino también de la energía que es cedida al centellador.

Ahora bien, como podemos notar, es el fotocátodo quien convierte la luz incidente en una corriente de electrones gracias al efecto fotoeléctrico, para facilitar el paso de esta luz, el material fotosensible es depositado en una capa delgada en el interior del fotomultiplicador que es usualmente hecho de vidrio o cuarzo. Teniendo en cuenta la formula de Einstein:

$$E = hv - \phi \tag{4.27}$$

donde E es la energía cinética del electrón emitido, v es la frecuencia de la luz incidente y ϕ es la función trabajo, es claro que para un cierto número de frecuencias es requerido que antes tenga lugar el efecto fotoeléctrico. Por encima de este umbral, además, la probabilidad para este efecto esta lejos de ser una unidad. Por supuesto, la eficiencia para conversiones fotoeléctricas varia fuertemente con la frecuencia de la luz incidente y la estructura del material. Esta respuesta espectral en general es expresada por la eficiencia cuántica $n(\lambda)$:

$$n(\lambda) = \frac{\text{número de fotoelectrones liberados}}{\text{número de fotones incidentes en el cátodo}(\lambda)} \tag{4.28}$$

donde λ es la longitud de onda de la luz incidente. Una cantidad equivalente es la sensibilidad del cátodo radiante que se define como:

$$E(\lambda) = \frac{I_k}{P(\lambda)} \tag{4.29}$$

donde I_k es la corriente de emisión fotoeléctrica desde el cátodo y $P(\lambda)$ es el poder de radiación incidente. La sensibilidad del cátodo radiante es usualmente dada en unidades de $Ampere/Watts$ y esta relacionada a la eficiencia cuántica con:

$$E(\lambda) = \lambda n(\lambda)\frac{e}{hc} \tag{4.30}$$

Para E en $[A/W]$ y λ en nanómetros:

$$E(\lambda) = \frac{\lambda n(\lambda)}{1240}[A/W] \tag{4.31}$$

Fotomultiplicadores de Silicio (SiPM)

MPPC es la abreviatura de contador de fotones de multi-pixel (Multi-Pixel Photon Counter por sus siglas en ingles), y este detector también se conoce como fotomultiplicador de silicio (SiPM). Es un fotodetector de estado sólido que utiliza múltiples píxeles de fotodiodo de avalancha (APD) que operan en modo Geiger[24].

El elemento básico (un píxel) de un MPPC es una combinación del modo Geiger APD y la resistencia de enfriamiento, y una gran cantidad de estos píxeles están conectados eléctricamente y dispuestos en dos dimensiones.

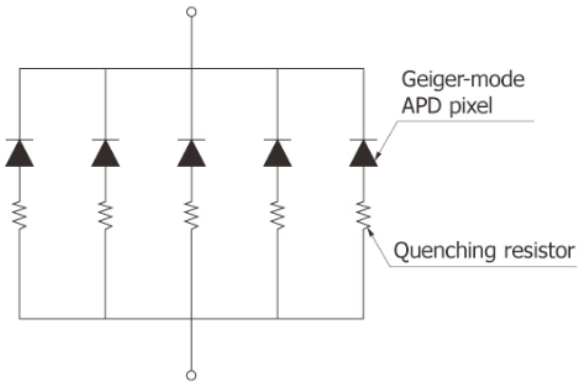

Figura 4.8: Estructura de un MPPC, combinación del modo Geiger APD y la resistencia de enfriamiento[25]

Cada píxel en el MPPC emite un pulso a la misma amplitud cuando detecta un fotón. Los pulsos generados por múltiples píxeles se emiten mientras se superponen entre sí. Por ejemplo, si tres fotones son incidentes en diferentes píxeles y detectados al mismo tiempo, entonces el MPPC emite una señal cuya amplitud es igual a la altura de los tres pulsos superpuestos. Cada píxel emite solo un pulso y esto no varía con el número de fotones incidentes. Por lo tanto, el número de pulsos de salida es siempre uno, independientemente de si un fotón o dos o más fotones entran en un píxel al mismo tiempo. Esto significa que la linealidad de salida MPPC empeora a medida que se producen más fotones en el MPPC, como cuando dos o más fotones entran en un píxel. Esto hace que sea esencial seleccionar un MPPC que tenga

suficientes píxeles para que coincida con el número de fotones incidentes. Para ello existen dos métodos que nos ayudaran a estimar el número de fotones que se detectan en el MPPC.

El primer método es la observación de pulsos, cuando la luz entra en un MPPC en un momento determinado, su altura de pulso de salida varía según el número de fotones detectados. La figura 4.9 muestra los pulsos de salida del MPPC obtenidos cuando se iluminó con la luz pulsada a niveles de conteo de fotones y luego se amplificó con un amplificador lineal y se observó en un osciloscopio. Como se puede ver en la figura 4.9, los pulsos se separan entre sí de acuerdo con el número de fotones detectados como uno, dos, tres fotones, etc. Medir la altura de cada pulso permite estimar el número de fotones detectados[24].

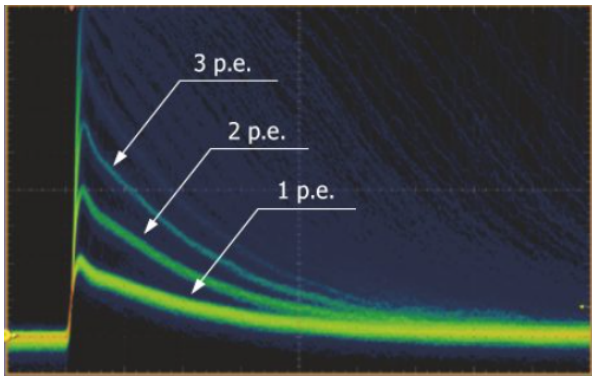

Figura 4.9: Vista de un osciloscopio de las formas de onda de pulso cuando se utiliza un amplificador lineal[25]

El segundo método de estimación de fotones es el de integración de la carga de salida, aquí la distribución del número de fotones detectados durante un período particular se puede estimar midiendo la carga de salida del MPPC utilizando un amplificador de carga o algún otro dispositivo similar. La figura 4.10 muestra una distribución obtenida discriminando el monto de carga acumulada. Cada pico de la izquierda corresponde al pedestal, un fotón, dos fotones, tres fotones, etc. Dado que la ganancia de MPPC es lo suficientemente alta como para producir una gran cantidad de carga de salida, la distribución puede mostrar picos discretos de acuerdo con el número de fotones detectados.

Las características de MPPC varían mucho dependiendo del voltaje de funcionamiento y la

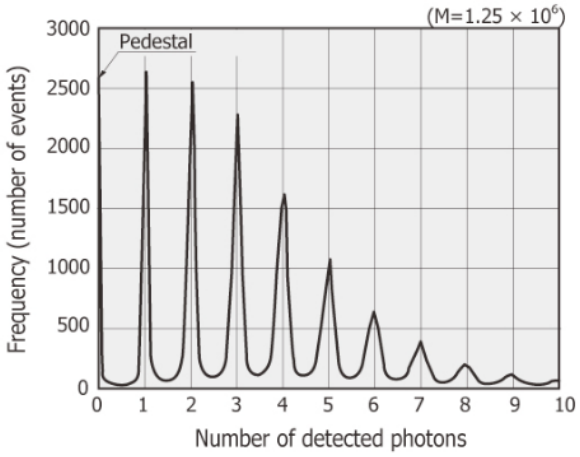

Figura 4.10: Espectro de altura de pulso cuando se utiliza el amplificador de carga[25]

temperatura ambiente. En general, elevar el voltaje de funcionamiento aumenta el campo eléctrico dentro del MPPC y, por lo tanto, mejora la ganancia, la eficiencia de detección de fotones y la resolución del tiempo. Por otro lado, esto también aumenta los componentes no deseados, como el recuento de ruido, los pulsos retardados y la diafonía (una perturbación electromagnética) que reducen el S/N. El voltaje de funcionamiento debe establecerse cuidadosamente para obtener las características deseadas. El MPPC se puede utilizar por varios métodos de acuerdo con la aplicación. Aquí presentamos un método típico para observar pulsos de luz. El uso de un amplificador de banda ancha y un osciloscopio facilita esta medición. La figura 4.11 muestra un ejemplo de una conexión a un amplificador de banda ancha. La resistencia de 1 $k\Omega$ y el condensador de 0,1 μF en la parte de la fuente de alimentación sirven como un filtro de paso bajo que elimina el ruido de alta frecuencia de la fuente de alimentación. La resistencia de 1 $k\Omega$ es también una resistencia protectora contra la corriente excesiva. El MPPC en sí es un detector de bajo nivel de luz, sin embargo, en los casos en que una gran cantidad de luz ingresa al MPPC, por ejemplo, cuando está acoplado a un centellador para detectar radiación, una gran corriente fluye hacia el MPPC. Esto puede causar una caída de voltaje significativa a través de la resistencia protectora, por lo que el valor de la resistencia protectora debe seleccionarse cuidadosamente de acuerdo con el tipo de uso que se este teniendo presente. El amplificador debe estar conectado lo más cerca posible del

MPPC.

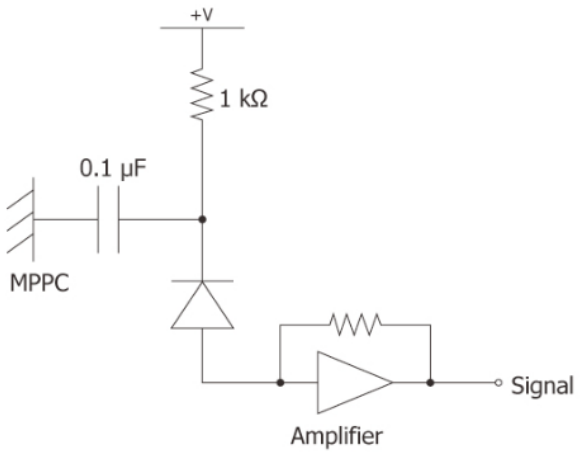

Figura 4.11: MPPC con conexión a un amplificador de banda ancha[25]

Algunas de las características que posee el MPPC son las siguientes, su ganancia de eficiencia cuántica esta cerca de los 10^6 por lo que se cataloga como un nivel de ganancia media con un voltaje de funcionamiento de 30 a 60 V para superficies grandes o relativamente del orden de $1m$, posee un multicanal con brecha estrecha y un circuito de lectura sencillo, ruido medio, una excelente uniformidad de lectura, a la vez de una alta resolución energética, con una sensibilidad media a la temperatura, inmunidad a la luz, resistencia magnética y en cuanto a diseño es compacto y de bajo peso.

Los fotodiodos de silicio PN (SiPN) y PIN son los fotodetectores de silicio más básicos que podemos encontrar actualmente en el mercado, su operación proviene de los cristales típicos con niveles de pureza más altos (equilibrio entre portadores de carga).Ya que la resistencia intrínseca del silicio es bastante alta este es ineficiente a la hora de la recolección de carga, los SiPN se fabrican formando una unión PN en silicio cristalino, las regiones P (ánodo) y N (cátodo) resultantes se conocen comúnmente como extrínsecas con una resistencia mucho menor que el silicio intrínseco, ya que los aumentos en la concentración de portadores mayoritarios disminuyen la resistencia eléctrica de una región dopada (cátodo). Dado que todos los átomos que participan en la red de un cristal semiconductor monolítico comparten la misma valencia y los niveles de energía de la banda de conducción, los electrones en la banda de

conducción y los huecos en la banda de valencia pueden moverse dentro de las dimensiones físicas del cristal. Sin embargo, cualquier tipo de portador puede beneficiarse de una movilidad mucho mayor bajo un campo eléctrico. desplazándose así rápidamente a través de un campo E que difundiéndose a través del cristal semiconductor mediante un paso aleatorio browniano en ausencia de un campo eléctrico. Una vez formada la union PN, un hueco y un electrón se recombinan para neutralizar la carga eléctrica de cada uno (Zona de deplexión o linea de unión). Dicha combinación agota la capa adyacente a la unión en cada lado de los portadores mayoritarios (por lo que se forma lo que se denomina como capa de agotamiento), por lo tanto, ahora los electrones del cátodo les costara más trabajo atravesar la linea de unión y así combinarse con un hueco ya que se interpuso una zona de deplexión que se opone a ello, sin embargo, existe la posibilidad de que algunos electrones logren pasar esta barrera y en ese momento la zona de deplexión se hará más ancha y al ser más ancha, entonces, más trabajo les costara a futuros electrones atravesar esta barrera de la linea de unión, de esta manera cuando ya no atraviesan esta linea ningún electrón proveniente del cátodo, se llegará al equilibrio figura 4.12 y entonces prácticamente la zona de deflexión se convierte en una barrera para los electrones libres del lado del cátodo. Si analizamos la zona de agotamiento de ambas partes tanto de P (ánodo) como de N (cátodo), nos damos cuenta de que el lado N queda cargado positivamente de la capa hacia su lado P que queda cargado negativamente, señalando así desde el cátodo hacia el ánodo, formando así una diferencia de potencial llamado muy comúnmente como potencial de barrera, por lo que, cualquier electrón que desee pasar a través de dicha unión PN, deberá vencer este potencial de barrera.

Sin embargo, si se puede usar una fuente externa de energía para generar un excedente de portadores de carga en cualquier lado de la región de agotamiento, por ejemplo por efecto fotoeléctrico, es posible alterar ese equilibrio e inducir una corriente neta a través de la unión, de manera que se pueda proporcionar suficiente energía a un electrón en la banda de valencia (N) para pasar a la banda de conducción (P) y dejar un hueco en la valencia figura 4.13. En este proceso, solo los portadores minoritarios (electrones generados en el lado P de la capa de agotamiento y huecos generados en el lado N de la capa) se desplazan a través de la unión PN para llegar a los bordes exteriores de la capa de agotamiento; portadores mayoritarios (los

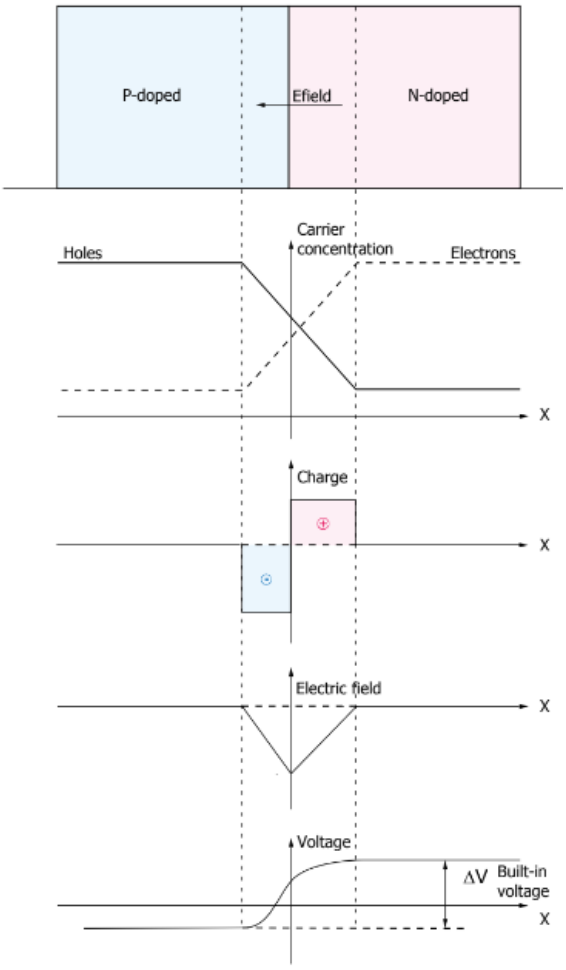

Figura 4.12: La unión PN en equilibrio[26]

electrones generados en el lado N y los huecos generados en el lado P) no cruzan la unión sino que se desplazan para alcanzar los bordes exteriores de la capa. Después de que el exceso de electrones y huecos alcanzan el borde exterior respectivo de cada capa de agotamiento, los electrones se acumulan en la región N mientras que los huecos se acumulan en la región P, si la capa de agotamiento está expuesta a fotones de luz de energías mayores que la brecha de la banda de silicio, que es 1,14 eV de diferencia de energía entre las bandas de conducción y valencia del silicio.

Debemos tener en cuenta que la profundidad de la capa de agotamiento afecta la fotosensibi-

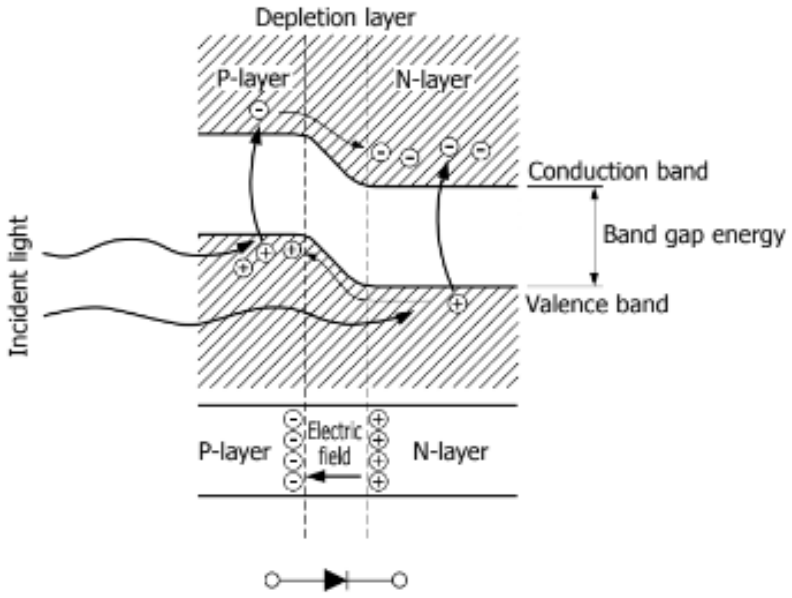

Figura 4.13: Visualización de la brecha de banda de silicio con una diferencia de energía entre las bandas de valencia y conducción[26]

lidad y la respuesta de frecuencia del fotodiodo PN, cuando se aumenta la profundidad de la capa de agotamiento aumenta la fotosensibilidad del fotodiodo PN figura 4.14 y disminuye la capacitancia de la unión PN, mejorando proporcionalmente el ancho de banda de frecuencia de mismo. En términos de aplicabilidad, la capa de agotamiento de un fotodiodo PN se puede profundizar polarizando la unión PN a la inversa, lo que significa polarizar la región N (cátodo) a un potencial eléctrico más alto que la región P (ánodo). También para mejorar aún más la respuesta de frecuencia y el ancho de banda de los fotodiodos PN a una región se le puede introducir silicio intrínseco entre las regiones P y N, formando así un fotodiodo PIN, esta región intrínseca da como resultado un fotodiodo de menor capacitancia por unidad de área y, por lo tanto, mayor frecuencia de corte y ancho de banda. Estos fotodiodos operan con una ganancia de 1 y, por lo tanto, son adecuados para detectar señales de luz relativamente fuertes.

Para explicar ahora como funciona el proceso de avalancha (APD) en el fotodiodo debemos aumentar el voltaje de polarización inversa a través de la unión PN, lo que aumenta la fuerza

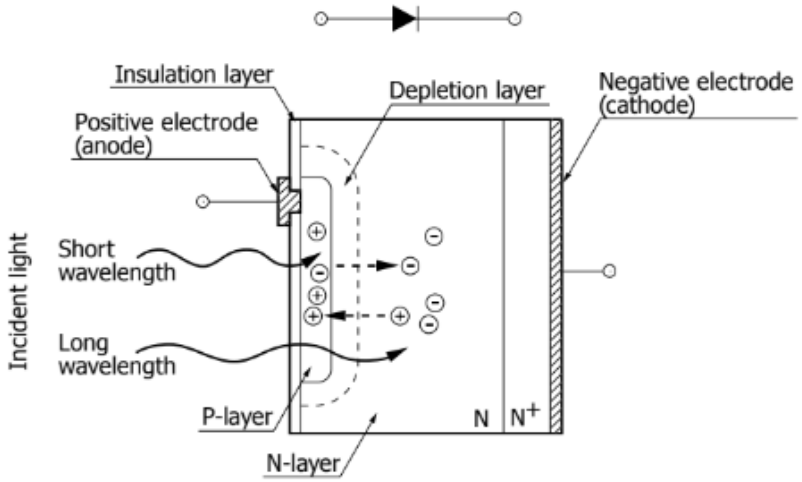

Figura 4.14: Absorción de longitudes de onda más cortas frente a longitudes de onda más largas en un fotodiodo SiPN[26]

eléctrica ejercida sobre los portadores de carga (electrones y huecos). Los portadores acelerados por esta fuerza experimentan un aumento en la velocidad y, por lo tanto, en la energía cinética entre colisiones de dispersión con los átomos de la red del cristal. Sin embargo, si la fuerza del campo E es tal que la energía media de los portadores excede la energía de la brecha de la banda de silicio, emergen portadores suficientemente energéticos que probablemente ionizan los átomos de la red tras el impacto y liberan al menos otro par de huecos de electrones en las bandas de conducción y valencia por cada impacto. Este efecto de ionización por impacto constituye un fenómeno de multiplicación de portadores asemejándose a una avalancha.

El único beneficio de la ganancia de avalancha es aumentar la magnitud de la señal fotoeléctrica original, sin embargo, eso a su vez contribuye con un factor de exceso de ruido a la señal original como una fluctuación aleatoria en la ganancia interna general del APD figura 4.15.

Dicha ganancia de la señal fotoeléctrica original del APD depende de la longitud de onda de la luz incidente (figuras 4.16 y 4.17) y aumenta con el voltaje de polarización inversa (figura 4.18). Además, la ganancia disminuye con la temperatura como se muestra en la figura 4.19. En el caso de la dependencia de la ganancia de la longitud de onda figura 4.16, podemos

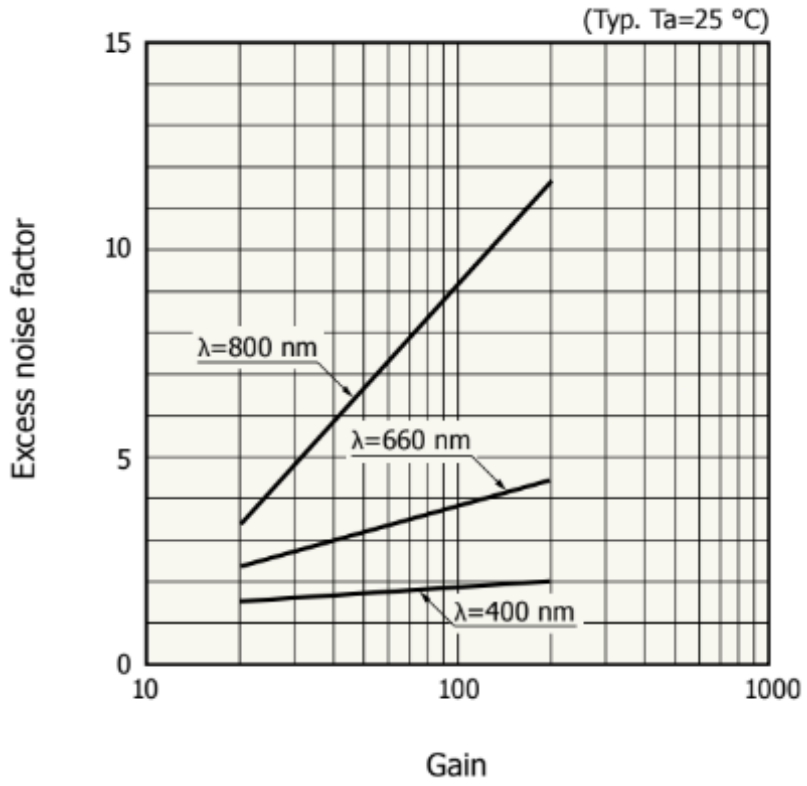

Figura 4.15: Gráficos del factor de exceso de ruido frente a la ganancia en tres longitudes de onda diferentes para un APD de Hamamatsu[26]

ver que un APD se puede fabricar en dos estructuras alternativas NP y PN. La estructura NP ofrece mayor ganancia y fotosensibilidad mejorada en respuesta a longitudes de onda más largas, mientras que la estructura PN tiene mayor ganancia y fotosensibilidad en respuesta a longitudes de onda más cortas. Esta diferencia en la estructura también se aplica al MPPC, pero solo afecta su fotosensibilidad. En la figura 4.17 vemos la dependencia de la ganancia de APD del voltaje inverso aplicado, en donde notamos tres comportamientos distintos, (A) por debajo de aproximadamente 50 V, (B) entre aproximadamente 50 V y aproximadamente 100 V, y (C) por encima de 100 V. Como sabemos la determinabilidad precisa de la ganancia de APD es fundamental para su aplicación en mediciones de alta precisión, por lo que una ganancia estable es un aspecto fundamental para mantener la linealidad de respuesta general del APD.

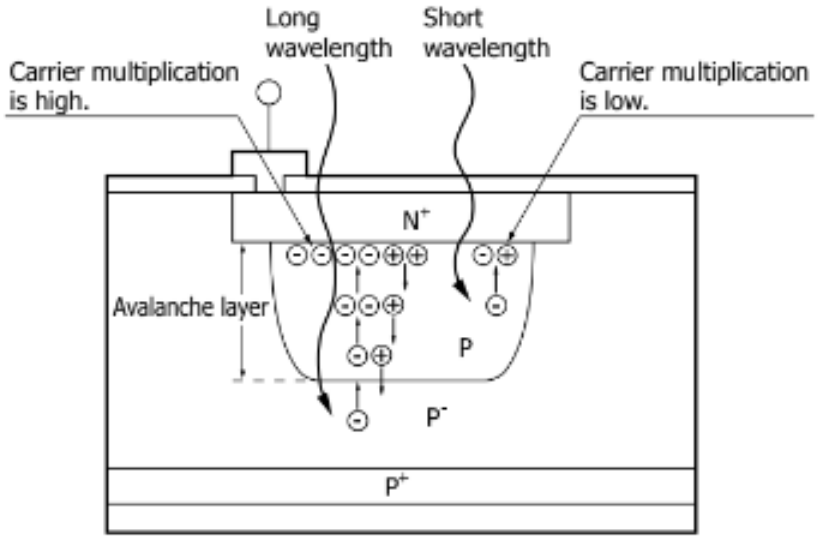

Figura 4.16: Visualización de la absorción de longitudes de onda de luz cortas y largas en una estructura APD mejorada[26]

Aunque una formulación matemática integral del proceso de avalancha es bastante compleja, para ilustrar esa linealidad de la manera más simple posible podemos caracterizar el proceso de avalancha como:

$$2 + 2P + 2P^2 + 2P^3 + ... + 2P^n = 2\sum_{i=0}^{n} P^i = 2\frac{1}{1-P} \tag{4.32}$$

Esto para $n \to \infty$. Ahora, si aumentamos el número inicial de fotoelectrones y agujeros que desencadenan la avalancha a 2m se tiene:

$$2m + 2mP + 2mP^2 + 2mP^3 + ... + 2mP^n = 2m\sum_{i=0}^{n} P^i = 2m\frac{1}{1-P} \tag{4.33}$$

Observamos así que la relación entre las dos poblaciones finales de avalancha es m, y por tanto, existe una relación lineal de proporcionalidad por un factor de m entre las dos poblaciones finales. Sin profundizar en su complicada matemática derivación, la ganancia de APD como factor de multiplicación de portadores se puede formular como:

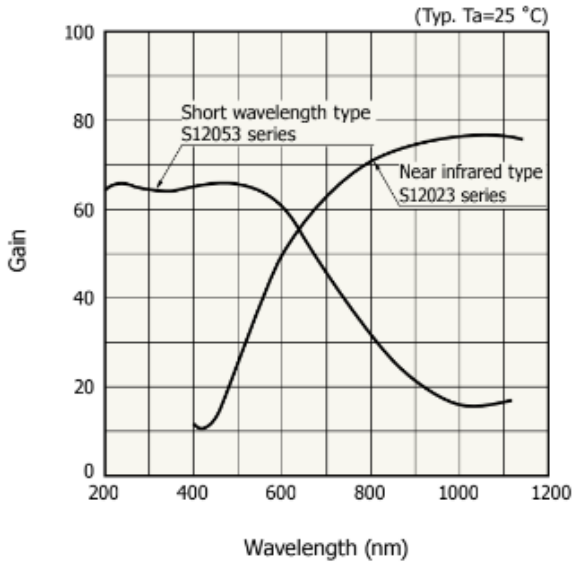

Figura 4.17: Dependencia obtenida de la longitud de onda de la luz incidente para dos APD de Hamamatsu[26]

$$M = \frac{(\alpha_e - \alpha_h).e^{d.(\alpha_e - \alpha_h)}}{\alpha_e - \alpha_h.e^{d.(\alpha_e - \alpha_h)}} = \frac{1-k}{e^{(k-1).\alpha_e.d} - k} \tag{4.34}$$

Donde α_e y α_h son las eficiencias de ionización (probabilidades por unidad de distancia recorrida) de avalanchas de electrones y huecos, k es la relación entre la eficiencia de ionización de un hueco y la de un electrón:

$$k = \frac{\alpha_h}{\alpha_e} \tag{4.35}$$

y d es el espesor de la capa de avalancha representada en la Figura 4.20

Un aspecto peculiar de la operación en modo Geiger es la densidad de población de las avalanchas En modo Geiger, esta densidad alcanza niveles tan altos que la nube de portadores en avalancha se comporta como un conductor con muy baja resistencia, esto resulta en una oleada de corriente a través del APD, lo que hace que el APD no pueda producir pulsos de salida discretos. por lo tanto, es necesario extinguir ese pico de corriente una vez que se hayan alcanzado la ganancia suficiente. La figura 4.21 se muestra el circuito equivalente de un APD en modo Geiger (GAPD) y se puede utilizar para describir cómo se realiza dicha

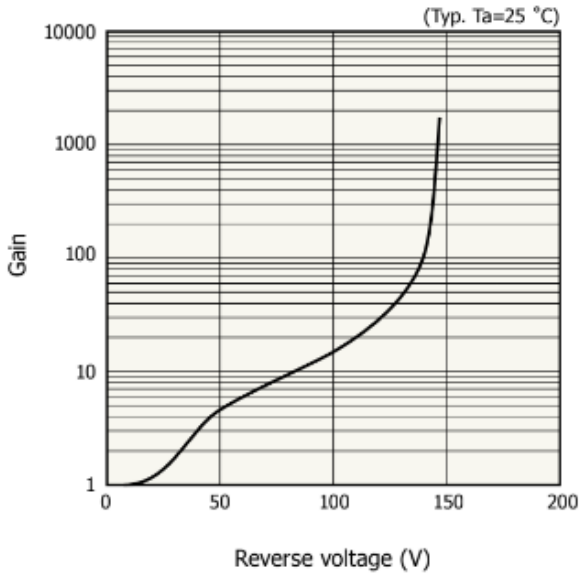

Figura 4.18: Gráfico de ganancia de APD frente a voltaje de polarización inversa[26]

extinción.

En este circuito equivalente, la capacitancia del GAPD (C_d) está polarizada inicialmente en V_{bias} mientras el interruptor se encuentra abierto. Una vez que se genera un par electrón-hueco dentro de la capa de agotamiento (ya sea térmicamente o por el efecto fotoeléctrico), el interruptor se cierra y C_d comienza a descargarse a través de la resistencia en serie (R_s) del GAPD, cuyo pequeño valor proporciona un aumento en el flujo de corriente mientras que la diferencia de potencial a través de C_d (llamémoslo V_d) decae exponencialmente hacia el voltaje de ruptura (V_{bd}). Esta disminución de V_d debilita el proceso de avalancha, aumentando R_s y disminuyendo el flujo de corriente de descarga de C_d. Con base en los dos bucles de corriente del circuito equivalente del GAPD, se puede establecer la siguiente relación para describir el flujo de corriente neto a través de C_d:

$$I_d = \frac{(V_d - V_{bd})}{R_s} + \frac{(V_{bias} - V_d)}{R_q} \tag{4.36}$$

Dado que los flujos de corriente en los dos bucles estarían en direcciones opuestas (uno ingresando a C_d para recargarlo y otro saliendo de C_d para descargarlo), la corriente neta I_d

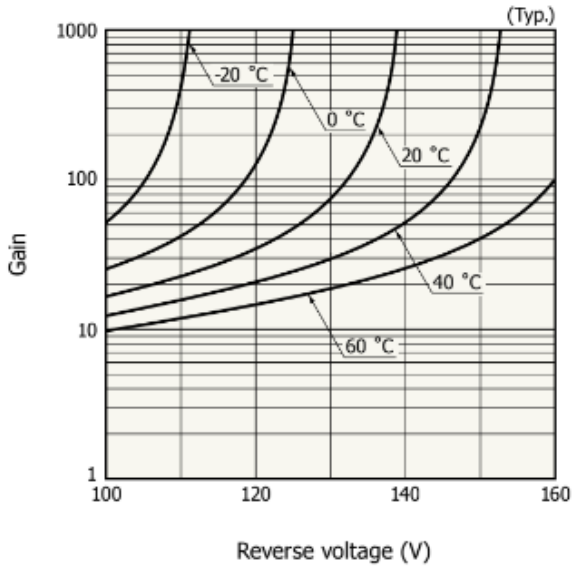

Figura 4.19: Efectos de la temperatura de ganancia frente al voltaje inverso[26]

fluiría en la dirección del flujo de corriente más fuerte. Así, si en el punto donde V_d es tal que:

$$V_d - V_{bd} = V_{bias} - V_d \tag{4.37}$$

la resistencia de extinción R_q es tan pequeña que continúa fluyendo una corriente de recarga hacia C_d y mantiene su descarga a través de R_s, el proceso de extinción no tendrá lugar. Sin embargo, si R_q es lo suficientemente grande como para que el flujo de corriente de V_{bias} no pueda sostener la descarga de C_d, el proceso de avalancha se extinguirá. Así, una vez que la descarga se agota y llega a su valle, el proceso de avalancha se apaga y se abre el interruptor conceptual. La corriente de recarga que fluye hacia Cd a través de Rq aumenta el voltaje a través de Cd en $V_{bias} - V_{bd}$ para igualar V_{bias} preparando el GAPD para la siguiente avalancha. Cabe señalar que los valores de R_q lo suficientemente grandes como para facilitar el proceso de extinción generalmente se determinan empíricamente. El proceso de recarga de C_d se denomina comúnmente recuperación y su duración se caracteriza generalmente por la constante de tiempo $R_q.C_d$. Durante el breve período de multiplicación de la avalancha (que

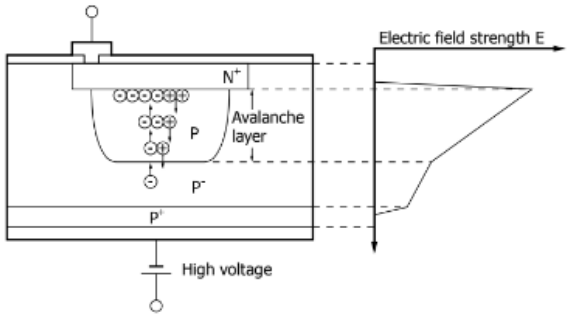

Figura 4.20: La región de avalancha dentro de un Si APD[26]

corresponde a la constante de tiempo rápido de $R_s.C_d$) y el tiempo de recuperación posterior, un GAPD no está disponible para detectar un nuevo evento fotoeléctrico; Para disminuir el tiempo de recuperación del GAPD, su capacitancia neta puede reducirse teóricamente introduciendo una capacitancia más pequeña en serie con Cd, este arreglo teórico se puede implementar introduciendo un filtro de paso alto con una constante de tiempo RC deseablemente más corta que el tiempo de recuperación de GAPD en la entrada del circuito amplificador utilizado para la lectura de GAPD Figura 4.21, el pulso de salida del amplificador tiene una forma tal que la pérdida de ganancia total neta es:

$$1 - \frac{\frac{f_{GAPD}}{f_c}}{\sqrt{1 + (\frac{f_{GAPD}}{f_c})^2}} \tag{4.38}$$

donde fc es la frecuencia de corte del filtro

$$\frac{1}{2\pi R_{filter}.C_{filter}} \tag{4.39}$$

y f_{GAPD} es el ancho de banda de frecuencia de la recuperación de GAPD

$$f_{GAPD} = \frac{1}{2\pi R_q.C_d} \tag{4.40}$$

mientras que el tiempo de caída del pulso de salida del amplificador se puede obtener como el RMS de los tiempos de caída del amplificador y del GAPD para ser:

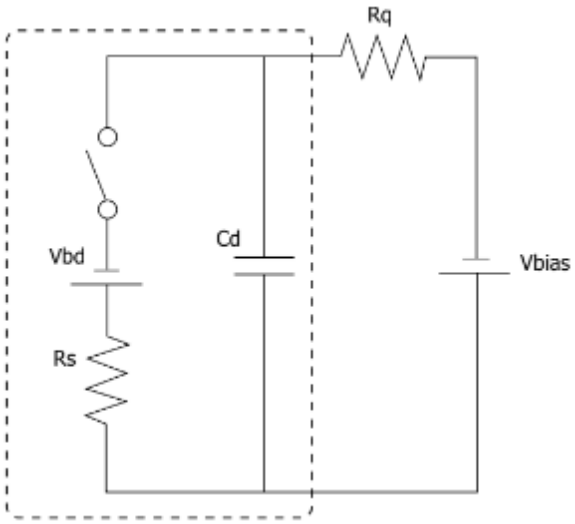

Figura 4.21: Circuito equivalente de un APD en modo Geiger (GAPD)[26]

$$2{,}2 * \sqrt{R_{filter}^2.C_{filter}^2 + R_q^2.C_d^2} \tag{4.41}$$

o también

$$0{,}35 * \sqrt{\frac{1}{f_c^2 + f_{GAPD}^2}} \tag{4.42}$$

Considerando $BW = 0{,}35/(aumento/tiempodecaida)$ y ese tiempo de caída de un circuito RC es de aproximadamente 2.2 veces su constante de tiempo. Los niveles extremadamente altos de ganancia de avalancha en el modo Geiger hacen que este régimen de operación APD sea particularmente interesante para aplicaciones con bajos niveles de luz.

4.5.2. Detección Indirecta

Este tipo de detección por otro lado se analiza el flujo de radiación cuyos valores de energía están por encima de los $10^{15}eV$, en general, el fenómeno que se presenta al aumentar la energía es que los rayos cósmicos tienden a ser más infrecuentes y por ende su detección se hace cada vez más compleja. En este caso las partículas que participan en los procesos y en

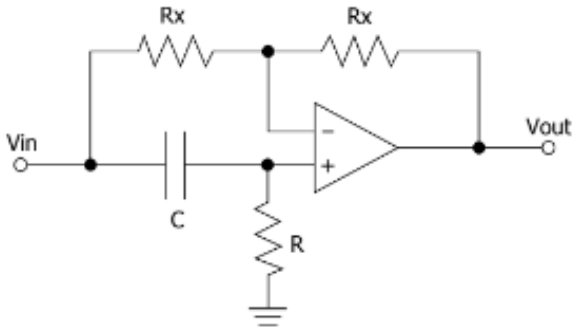

Figura 4.22: Esquema de un circuito amplificador de paso alto V/V[26]

general a muy altas energías ($> 10^{15} eV$) producen otras partículas que se detectan cuando interactúan o chocan con los núcleos de otros átomos, haciendo posible así detectar partículas a partir de mecanismos como por ejemplo los detectores de fluorescencia y de superficie que aprovechan la fluorescencia producida por la luz ultravioleta producto del choque de moléculas con el aire excitado. Prueba de ellos se encuentra en el observatorio Pierre Auger ubicado en el hemisferio sur, en los departamentos de Malargüe y San Rafael, provincia de Mendoza, República Argentina[27].

Capítulo 5

Física óptica

Cuando hablamos de física óptica nos referimos a aquellos procesos ópticos que relacionan la interacción entre materia y radiación electromagnética estos incluyen generación de luz de centelleo, la emisión de radiación Cherenkov, absorción masiva, dispersión de Rayleigh, procesos de contorno y procesos de guías de luz o fibras ópticas (WLS por sus siglas en ingles).

5.1. Centelleo

El centelleo es un termino general para referirse al proceso que hace uso de la propiedad que tienen ciertos materiales de emitir un pequeño rayo de luz cuando son golpeados por una partícula o radiación nuclear, puede liberar fotones en el rango de luz visible o ultravioleta (UV), cuando un electrón es excitado en el centellador vuelve a su estado fundamental y estos fotones de centello serán detectados por un fotomultiplicador que se encargara luego de transformar la luz en señal eléctrica que luego se amplifica aun más con un sistema de multiplicación de electrones.

Los materiales de centelleo poseen la propiedad que se conoce como luminiscencia, cuando estos materiales son expuestos a cierta forma de energía, por ejemplo, luz, radiación, calor, entre otras, estos absorben y reemiten energía en forma de luz visible. Si la reemisión ocurre inmediatamente después de la absorción o más precisamente con $10^{-8}s$ (que es el tiempo en que toma una transición atómica) el proceso es usualmente llamado fluorescencia. Sin embargo, si la reemisión es demorada porque el estado excitado es metaestable , el proceso es llamado fosforescencia o afterglow. En esos casos, el tiempo de demora entre absorción y reemisión puede durar desde unos cuantos microsegundos a horas dependiendo del tipo de

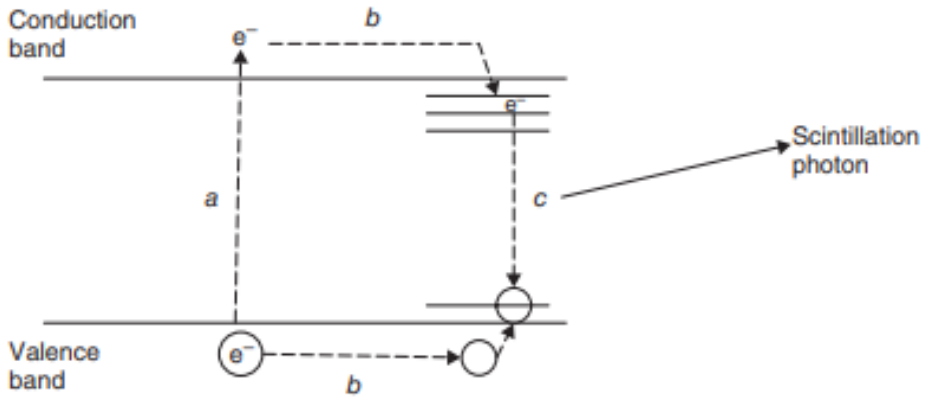

Figura 5.1: Estructura de banda y proceso de centelleo en talio y yoduro de sodio activado, los tres pasos en el proceso de centelleo son: a) Algunos de los electrones libres creados por interacción de rayos gamma tienen suficiente energía como para saltar a través de la brecha prohibida de la banda de conducción. b) Una vez en la banda de conducción, los electrones migran al centro de activación de los estados excitados de los orbitales y los agujeros en la banda de valencia para la activación de los orbitales en el centro del estado base. c) La transición de un electrón desde el centro de activación de orbitales en el estado excitado hasta el centro de activación orbital del estado base produce un fotón de centelleo.[28]

material.

Una primera aproximación para la evolución temporal del proceso de reemisión es descrito por un simple decaimiento exponencial, figura 5.2:

$$N = \frac{N_0}{\tau_d} exp\left(\frac{-t}{\tau_d}\right) \tag{5.43}$$

donde N es el número de fotones emitidos en un tiempo t, N_0 es el número total de fotones emitidos y τ_d es la constante de decaimiento. El aumento finito del tiempo desde cero al máximo en algunos materiales es usualmente mucho más rápido que el decaimiento temporal y se ha tomado aquí como cero por simplicidad.

Mientas esta simple representación es adecuada para la mayoría de las propuestas, algunas veces, se puede experimentar un decaimiento más complejo. Una descripción más precisa, en esos casos, estaría dada por una doble componente exponencial, así:

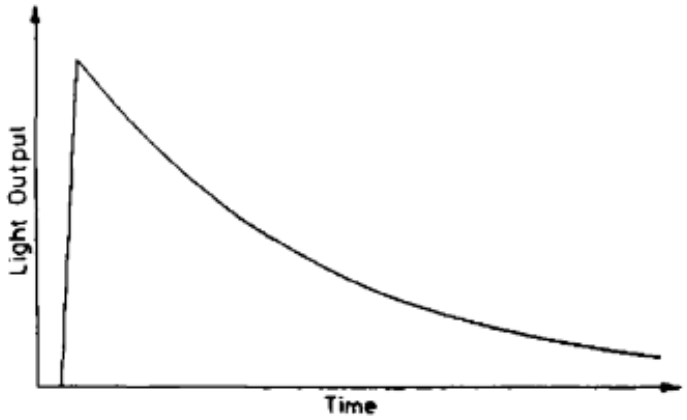

Figura 5.2: Decaimiento exponencial simple de radiación fluorescente. El aumento temporal es usualmente mucho más rápido que el decaimiento temporal[29]

$$N = Aexp\left(\frac{-t}{\tau_f}\right) + Bexp\left(\frac{-t}{\tau_s}\right) \tag{5.44}$$

donde τ_f y τ_s son las constantes de decaimiento. Para la mayoría de los centelladores una componente es generalmente mucho más rápida que otra por lo que es costumbre referirse a ellas como la componente rápida (fast, con subíndice f) y la componente lenta (slow,con subíndice s) o también como la componente inmediata y la componente demorada. Sus magnitudes relativas A y B, varían de material en material, y de ellas es generalmente la componente rápida (f) la que domina. La figura 5.3 muestra la relación entre estas componentes. Actualmente existen seis tipos de materiales de centelleo que son usados en gran medida: cristales orgánicos, líquidos orgánicos, plásticos, cristales inorgánicos, gases y lentes.

5.2. Radiación Cherenkov

La radiación Cherenkov surge cuando una partícula cargada en un medio material se mueve más rápido que la velocidad de la luz en ese mismo medio. Esta velocidad esta dada por:

$$\beta c = v = c/n \tag{5.45}$$

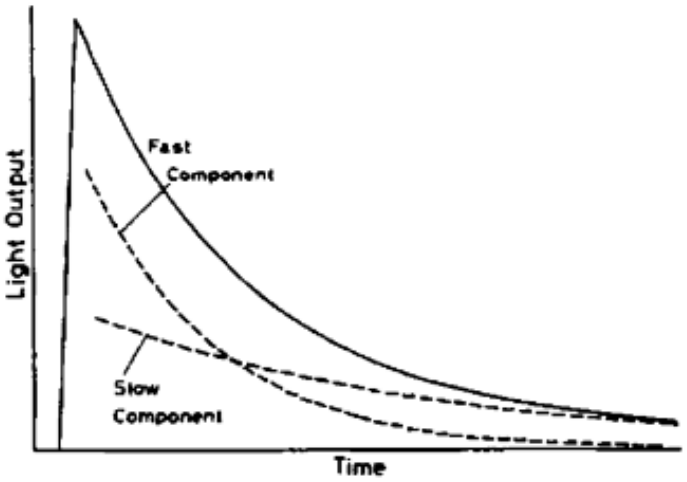

Figura 5.3: Respuesta de luz de centelleo entre componentes rápida y lenta. La linea solida representa la curva de decaimiento de luz total[29]

donde n es el índice de refracción del material y c es la velocidad de la luz en el vació. Una partícula emitiendo radiación Cherencov debe por lo tanto tener una velocidad:

$$v_{part} > c/n \tag{5.46}$$

En cuyo caso, una onda de choque electromagnética es creada, como un avión que va más rápido que el sonido, este crea una una onda de choque sónica, como se ve en la figura 5.4.

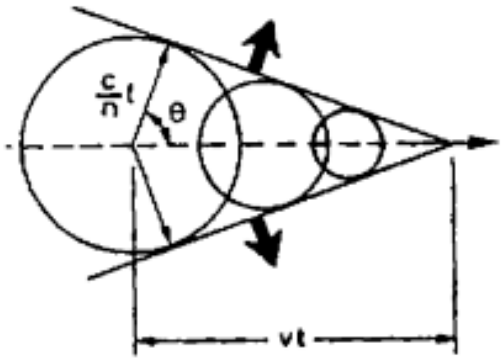

Figura 5.4: Radiación Cherenkov: una onda de choque electromagnética es formada cuando la partícula viaja más rápido que la velocidad de la luz en ese medio.

Vemos que el frente de onda formado es cónico en su forma y es emitido a un ángulo:

$$cos\theta = 1/\beta n \tag{5.47}$$

con respecto a la trayectoria de la partícula. En general, un espectro continuo de frecuencias es radiado con los fotones siendo linealmente polarizados. La energía transportada por radiación Cherenkov fue primero calculada por Tamm y Frank[30] siendo:

$$-\frac{dE}{dx} = \frac{4\pi e^2}{c^2}\int wdw\left(1 - \frac{1}{\beta^2 n^2}\right) \tag{5.48}$$

donde la integración es solamente para aquellas frecuencias para las cuales $\beta n(w) > 1$. esta energía perdida esta ya incluida en la formula de Bethe-Bloch (4.49) y es mucho mayor a velocidades relativistas. Incluso a estas energías, sin embargo, su contribución es pequeña en comparación con la pérdida por colisión. De hecho, para materiales condensados:

$$-\left(\frac{dE}{dx}\right)_c \approx 10^{-3} MeV cm^2 g^{-1} \tag{5.49}$$

El cual es despreciable con respecto a la perdida por colisión. Para gases como H_2:

$$-\frac{dE}{dx} \approx 0{,}1 MeV cm^2 g^{-1} \tag{5.50}$$

Mientras que para He y gases con mayor Z (número atómico):

$$-\frac{dE}{dx} \approx 0{,}01 MeV cm^2 g^{-1} \tag{5.51}$$

La dependencia del ángulo de emisión de radiación Cherenkov sobre la velocidad de la partícula ha sido particularmente estudiado por físicos de partículas en la forma de contadores Cherenkov. Tales dispositivos proporcionan una medición más precisa de las velocidades de las partículas.

5.3. Coeficiente de Absorción

La medida de la absorción de luz es una de las técnicas más importantes para las medidas ópticas en solidos. En la medida de adsorción, aquí se pone atención a la intensidad de luz $I(z)$ luego de recorrer cierta distancia z de material en comparación con la intensided insidente I_0 de este modo se define el coeficiente de absorción $\alpha_{abs}(w)$ como siendo:

$$I(z) = I_0 e^{-\alpha_{abs}(w)z} \tag{5.52}$$

La constante de absorción la podemos ver esquemáticamente en la figura 5.5. Donde la intensidad $I(z)$ depende del cuadrado de las variables de campo[28], tal que:

$$\alpha_{abs}(w) = 2\frac{w\tilde{k}(w)}{c} \tag{5.53}$$

Donde, w es la frecuencia de la luz, el factor 2 resulta de la definición de $\alpha_{abs}(w)$ en términos de la intensidad de luz, el cual es proporcional al cuadrado de los campos. Esta ecuación nos dice que el coeficiente de absorción es proporcional a $\tilde{k}(w)$, la parte imaginaria del índice de refracción complejo (coeficiente de extinción), ya que, $\tilde{k}$ es usualmente asociado con perdidas de potencia. Notemos que la ecuación (5.53) se aplica a la absorción del portador libre en semiconductores en el límite $w\tau \gg 1$ y $w \gg w_p$. Con τ siendo el tiempo de relajación de la partícula y w_p la frecuencia plasma del material.

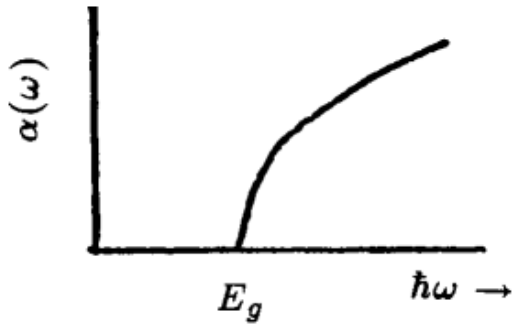

Figura 5.5: dependencia de la frecuencia del coeficiente de absorción cerca de un umbral para las transiciones entre bandas.[28]

Ahora mostraremos que la dependencia de la frecuencia del coeficiente de absorción es bastante diferente para los diversos procesos físicos que ocurren en las propiedades ópticas de los sólidos. Consideraremos aquí la dependencia de la frecuencia del coeficiente de absorción para:

1. Absorción de portadores libres
    - Semiconductores típicos $\alpha_{abs}(w) \sim w^{-2}$
    - Metales a bajas frecuencias $\alpha_{abs}(w) \sim w^{1/2}$
2. Transiciones directas entre bandas
    - Forma del coeficiente de absorción $\alpha_{abs}(w) \sim \frac{(\hbar w - E_g)^{1/2}}{\hbar w - E_g}$
    - Conservación del momento del cristal.
    - Relación entre m y el elemento de la matriz de momento.
    - Forma de $\alpha_{abs}(w)$ para transición directa prohibida $\sim \frac{(\hbar w - E_g)^{3/2}}{\hbar w - E_g}$
3. Transiciones indirectas entre bandas
    - Forma del coeficiente de absorción $\alpha_{abs}(w) \sim (\hbar w - E_g \pm \hbar w_q)^2$.
    - Procesos de absorción y emisión de fonones.

El resumen dado arriba es para sistemas 3D. En el caso de sistemas 2D y 1D, la dependencia funcional es sensible a la dimensionalidad del sistema para cada proceso.

5.4. Dispersión de Rayleigh

Las técnicas de dispersión de luz proporcionan una herramienta sumamente útil para estudiar excitaciones fundamentales en sólidos, como los fonones, porque la luz puede dispersarse de los sólidos de manera inelástica por lo que los fotones incidentes y dispersos tienen diferentes frecuencias. La dispersión de luz inelástica se convirtió en una herramienta importante para el estudio de excitaciones en sólidos a mediados de la década de 1960 con el advenimiento de

las fuentes de luz láser, porque la luz dispersada inelásticamente es típicamente solo $\sim 10^{-7}$ de la intensidad de la luz incidente.

Mucho antes de la mecánica cuántica, Lord Rayleigh (1871) analizó el esparcimiento de la luz solar tomando como referencia los osciladores moleculares. Utilizando un argumento simple basado en el análisis dimensional, Rayleigh llego a la correcta conclusión de que la intensidad de la luz esparcida era proporcional a $1/\lambda^4$ que es la longitud de onda o v^4 aumentando la frecuencia[31]. Antes de conocer ese trabajo se creía que el cielo era azul por el esparcimiento de las de minúsculas partículas de polvo. A partir de ese momento, el esparcimiento que implica partículas más pequeñas que una longitud de onda (es decir, aproximadamente menor a $\lambda/15$) se denomina esparcimiento Rayleigh. Los átomos y las moléculas normales reúnen esas condiciones ya que su diámetro alcanza unas cuantas décimas de nanómetro. La luz tiene aproximadamente una longitud de onda de unos 500nm. La formula analítica de la dispersión de Rayleigh esta dada por la formula:

$$\frac{dI_1(\alpha)}{d\Omega} = P_0 4\pi^2 \frac{(n_0-1)^2}{N_0^2\lambda^4}(cos(\theta)^2 cos(\alpha)^2 + sin(\theta)^2) \tag{5.54}$$

donde $\frac{dI_1(\alpha)}{d\Omega}$ es la intensidad de la dispersión de la luz por partícula, por unidad del ángulo solido Ω en la dirección α, P_0 es la potencia de la luz incidente por unidad de superficie, n_0 es el índice de refracción, N_0 es el número de partículas (moléculas) por unidad de volumen, λ longitud de onda, θ el ángulo entre el plano de dispersión y la dirección de polarización de la luz incidente.

Cualquier luz incidente sobre una partícula de diámetro $d \ll \lambda$ actúa como un campo eléctrico E_0 sobre esa partícula, causando polarización e induciendo un momento dipolar p, la fuerza de que depende de la polarización ϕ de la partícula:

$$p = \phi E_0 \tag{5.55}$$

Dado que la luz incidente es una onda y, por lo tanto, aplica un campo eléctrico incidente oscilante, el momento dipolar también oscila, lo que genera una radiación en todas las direcciones. En el caso más común, la polarización isotrópica, las direcciones del campo incidente

y el momento dipolar inducido coincide. En este caso, para un punto a la distancia $r \gg \lambda$, la intensidad de la luz dispersada se vuelve:

$$I = \frac{(1 + cos^2\theta)k^4|\alpha|^2}{2r^2} I_0 \tag{5.56}$$

donde θ es el ángulo de dispersión, I_0 la intensidad de la onda incidente y k es el vector de onda dado por $k = 2\pi/\lambda$. La dependencia con $1/\lambda^4$ es evidente y cualquier desviación de esa dependencia sugiere una contaminación del medio medido con partículas que no cumplen el criterio $d \ll \lambda$.

5.5. Cambio de longitud de onda (WLS)

Los materiales WLS absorben fotones y luego los vuelven a emitir con una longitud de onda (λ) más larga. Este efecto se conoce como fotoluminiscencia, que es una de las dos subcategorías de la fluorescencia. (la otra subcategoría es el centelleo). La figura 5.6 ilustra el principio de la fluorescencia. La absorción del fotón inicial excita un electrón a un subnivel vibratorio (1, 2,...) de un estado de mayor energía (S1). El electrón primero se desexcita de forma no radiativa a la vibración del nivel básico del estado de energía excitado (S1). Luego emite un fotón cuando se desexcita al estado fundamental energético (S0), generalmente a un subnivel vibratorio excitado. Esta es la razón, por qué el material es (parcialmente) transparente a su propia luz emitida[32].

Las principales propiedades que caracterizan un material WLS son el espectro de absorción y emisión, el tiempo de subida y bajada y el factor de multiplicación. El espectro de absorción especifica, la magnitud de la absorción de los fotones incidentes, la energía/longitud del fotón emitido se distribuye de acuerdo con el espectro de emisión y la distribución temporal de la emisión de fotones que se caracteriza por un aumento rápido y la suma de uno o más decaimientos exponenciales más lentos con diferentes tiempos de decaimiento (debido a diferentes trayectorias de decaimiento). El factor de multiplicación es el número de fotones emitidos por fotón absorbido, que suele ser 1 para los materiales WLS comerciales.

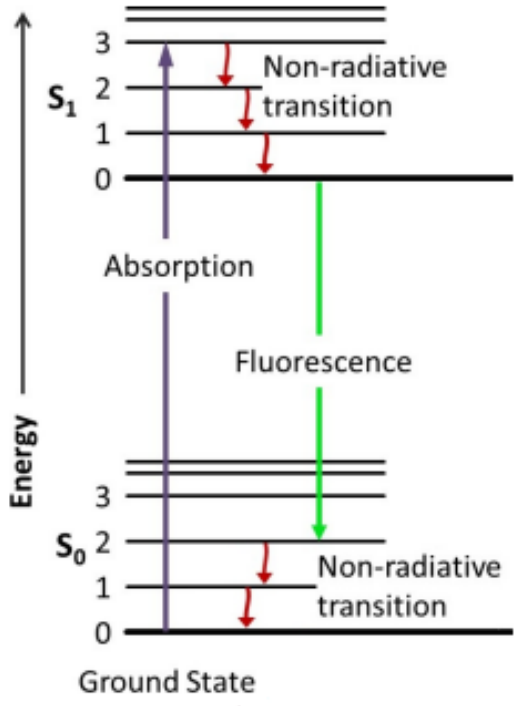

Figura 5.6: Diagrama de energía del proceso de fluorescencia[32]

En los detectores de partículas, los materiales WLS se utilizan para dos propósitos diferentes; por un lado, se pueden mezclar con materiales centelleantes para mejorar las propiedades del centellador. Además, se pueden usar en forma de fibras o barras para recolectar la luz de centelleo, cambiarla a longitudes de onda más apropiadas para el fotodetector utilizado, y guiarlo hacia el fotodetector.

Capítulo 6

GEANT4

Una simulación detallada de los procesos de la física óptica es necesario (en combinación con una simulación detallada de física de partículas). En este capítulo, se abordará el tema de la simulación computacional en física, además del marco de simulación de la física de partículas y sus propiedades ópticas con la ayuda del software GEANT4[3], que se ha utilizado en el ámbito de esta tesis, se presentará en detalle. En general, las simulaciones resultan muy útiles si las condiciones de ligadura de un modelo matemático básico de un problema son conocidas, mientras que al mismo tiempo una solución analítica del mismo problema es muy compleja o no es posible (con un esfuerzo razonable). Ejemplos típicos para la aplicación de tales simulaciones son del tipo Monte Carlo (MC), que depende del muestreo aleatorio repetido y análisis estadístico para calcular los resultados, ejemplo de ello son la determinación numérica de integrales o el análisis o predicción de la dinámica de fluidos (por ejemplo, corrientes marinas o corrientes de aire), reacciones termodinámicas, interacciones de partículas de alta energía dentro de materiales complejos (por ejemplo, en detectores de partículas o tejido humano), o el rendimiento de sistemas ópticos complejos (por ejemplo, volúmenes de centellador con lectura a través de fibras WLS), este método de simulación está muy relacionado a experimentos aleatorios, experimentos para los cuales el resultado específico no se conoce de antemano. En este contexto, la simulación de Monte Carlo se puede considerar como una forma metódica de hacer el llamado análisis hipotético. En el campo de la física de partículas, las simulaciones de MC se utilizan generalmente con dos propósitos. El primer propósito es el análisis de los datos medidos de un experimento. Aquí, las simulaciones de MC pueden servir como una predicción precisa de eventos de fondo. Esto es necesario para definir variables que separan los eventos de señal de los eventos de fondo y así interpretar los resultados del experimento. Por ejemplo, los análisis de datos que condujeron al reciente descubrimiento del bosón de Higgs [33] incluía miles de millones

de colisiones protón-protón simuladas y su firma en los diferentes detectores. Esto permitió determinar la significancia de la desviación entre la medición y la predicción del modelo estándar (representado por simulaciones MC). En contraste con esto, no se ha encontrado evidencia de la existencia del bosón de Higgs en experimentos anteriores y, por lo tanto, se han establecido límites en su masa (usando MC). El otro propósito para la aplicación de simulaciones MC es el diseño de detectores. En el ámbito del diseño u optimización de un detector de partículas, las simulaciones son una herramienta indispensable. Solo el marco de simulación multipropósito GEANT4 contiene una simulación de física óptica muy extensa y, a la vez, flexible, es por ende que traemos a colación dicha herramienta, basada en el lenguaje de programación C++, pues resulta sencilla y muy confiable para una simulación del paso de partículas a través de la materia y las interacciones que experimentan. Lo importante a resaltar aquí es que en el mismo entorno de la simulación es posible asignar una gran variedad de propiedades físicas que poseen los materiales de nuestro interés, en este caso la física óptica juega un papel importante para el desarrollo, el diseño y eficiencia de un detector de partículas.

6.1. Principios Básicos

GEANT4 es uno de los primeros intentos exitosos de rediseñar un importante paquete de software del CERN para la próxima generación de experimentos HEP (High Energy Physics por sus siglas en ingles) utilizando un entorno orientado a objetos. GEANT4 al ser un software basado en el lenguaje de programación C++, hace bastante uso de las denominadas $classes$ que hacen parte de un concepto más amplio de lo que comúnmente es llamado estructuras de datos, los cuales son un grupo de elementos de datos que son contenidos bajo un mismo nombre, donde dichos elementos pueden ser variables y/o funciones. Es por tanto que estas $classes$ constituyen todos lo aspectos considerados para desarrollar una simulación especifica y dentro del marco de la estructura que maneja GEANT4 existen las llamadas categorías de clases que son un conjunto de grupos de clases que mantiene una unión estrecha entre las mismas, pero están débilmente acoplados en relación con otros grupos, es decir,

que una categoría de clase contiene clases que tienen una relación cercana entre ellas[3]. Las categorías de clase y sus relaciones se presentan mediante el siguiente diagrama, Figura 6.1:

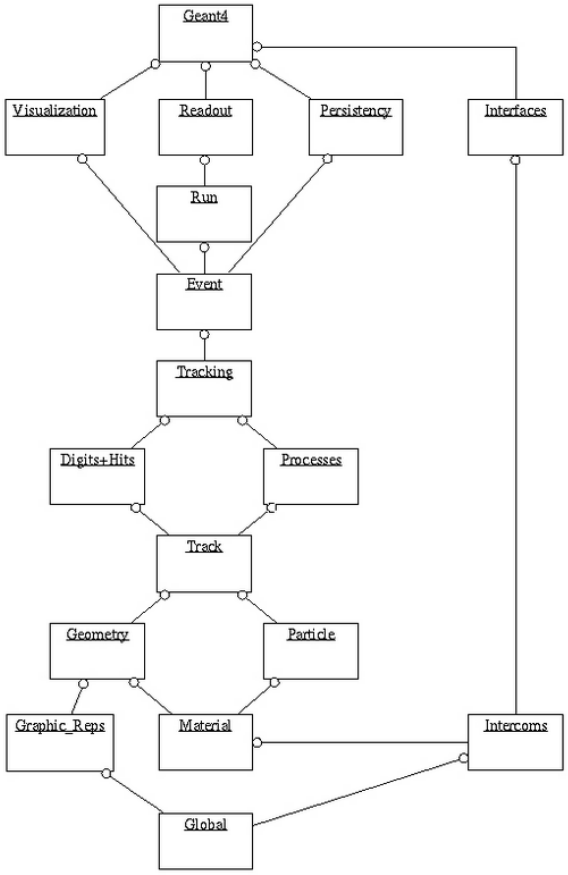

Figura 6.1: Categorías de clases en GEANT4[3]

Cada cuadro en la figura representa una categoría de clase y una relación de usos por una línea recta. El círculo al final de una línea recta significa que la categoría de clase que tiene este círculo usa la otra categoría.
Los aspectos que se consideran para ser utilizados dentro de una simulación en GEANT4 son los siguientes:

- Geometría del detector
- Materiales y sus propiedades

- Partículas elementales
- Generación de eventos primarios
- Procesos físicos que causan las interacciones
- Seguimiento de las partículas atravesando el material
- Componente sensitiva del detector
- Generación de datos de la simulación
- Visualización de geometrías y trayectorias de partículas

De esta forma, vemos como GEANT4 puede cubrir muchos procesos físicos para la física electromagnética y hadrónica (incluida la física óptica y de descomposición, así como la excitación y desexcitación atómica) para un amplio rango de energías desde la escala de electrón-voltios (eV) a teraelectrón-voltios (TeV) y para un rango espacial desde lo cósmico hasta la escala microscópica.

6.2. Simulación

GEANT4 no nos proporciona como tal un programa principal ($main()$), en cambio se debe crear dicho programa principal, este debe ser ejecutable y además debe combinar las $classes$ (extendidas) de GEANT4 y los controles de la simulación. Para crear una simulación de trabajo el usuario tiene que definir tres $classes$ obligatorias[3]:

- G4VUserDetectorConstruction, aquí se construyen todos los materiales, volumenes, geometria y detector sensitivo que se ajústa a los volúmenes del detector necesarios para la simulación,
- G4VUserPhysicsList, como GEANT4 no tiene partículas o procesos predeterminados, incluso para el transporte de partículas, se deben definir explícitamente. Esta clase

define todas las partículas necesarias para la simulación, así como los procesos y sus respectivas propiedades, valores y rangos de corte.

- G4VUserPrimaryGeneratorAction, para la generación de eventos primarios, esta clase es la que se encarga de generar los vértices primarios y las partículas primarias dentro de la simulación. En esta clase se combinan una o más partículas primarias, que se manejan una tras otra. Cada partícula primaria se crea en la posición y con las propiedades que ha especificado el usuario. Las partículas primarias y las partículas secundarias potenciales de las interacciones se rastrean a través del espacio y el tiempo en su camino a través del volumen simulado. El seguimiento de una partícula se detiene, si sale del volumen o si se detiene por la pérdida total de su energía cinética, por decaimiento o por una acción definida por el usuario.

En este punto cabe destacar la existencia de algunas clases que nos permiten realizar un análisis correspondiente de la interacción de las partículas, por lo general esta información se almacena el archivos con formato $.ROOT$, lo que nos permite manejar los resultados utilizando las siguientes $clases$:

- G4UserSteppingAction, usada para estudiar el recorrido de partículas y su energía total perdida en el proceso de su paso a través de la materia.
- G4AnalysisManager, usada para extraer la información, o datos, de la simulación modelada en un tipo de formato conveniente, en nuestro caso sera $.ROOT$ para el respectivo análisis.

6.2.1. Geometría y materiales

Ya que GEANT4 es una herramienta de simulación orientada a objetos, la definición de geometrías y aquellos materiales que las componen deben ser configurados en primera instancia a la hora de realizar cualquier simulación en curso. Dentro de este marco de programación existen algunas reglas generales a tener en cuenta. En primer lugar, se debe definir la forma

geométrica de un componente, el llamado ”sólido”. Por lo tanto, el usuario puede elegir desde objetos geométricos simples como cajas o cilindros hasta objetos más complejos como polígonos o sólidos rotacionales[34].

Lo siguiente que se debe hacer es asignar un material al sólido, para ello existen dos formas de implementar materiales en la simulación. La primera es hacer uso de las bibliotecas que vienen consigo dentro del marco GEANT4, ya que este admite la extracción de datos de algunos materiales predefinidos de las bases de datos del Instituto Nacional de Estándares y Tecnología (NIST por sus siglas en ingles), que contienen una gran cantidad de elementos, isótopos y compuestos químicos. Y la segunda forma en que se le puede asignar un material a un solido determinado es creando manualmente los materiales que se desean utilizar, para ello se debe especificar algunas de sus propiedades fundamentales para que dicho material se comporte como lo haría en la realidad, así como su composición de elementos o isótopos, su densidad y, si se requiere, su estado, presión o temperatura. Esto permite la máxima flexibilidad en la composición química del material. Ahora bien, en este caso nuestro interés esta en considerar procesos de física óptica dentro de la simulación, es por tanto necesario determinar y asignar dichas propiedades ópticas a los materiales que utilizaremos. Dicho proceso debe ser realizado por el usuario, tanto para los materiales introducidos de forma manual, así como para los materiales ya predefinidos por GEANT4, todos estos materiales necesitan al menos un espectro de índice de refracción (el cual corresponde a la relación de dispersión) y un espectro de longitud de atenuación, aunque la longitud de atenuación se establece en infinito si no se define, siendo ese su valor por defecto. Cuando introducimos materiales ópticos especiales, es decir, materiales centelleantes y WLS, requieren además la especificación de los espectros de emisión, así como de los tiempos de subida y bajada. Los espectros deben definirse como una función de la energía de la partícula y deben ordenarse por valores de energía crecientes. Para valores de energía entre dos puntos dados, los valores de un espectro de emisión se “interpolan” aplicando la media de los valores de las cantidades dependientes de la energía de esos puntos.

Cuando ya se tengan definidos los materiales, es momento de posicionar los volúmenes con los que se vaya a trabajar. Para ello, el usuario debe especificar su volumen madre (el volumen

en el que se van a colocar todos los demás volúmenes), su posición en el interior y su rotación con respecto al volumen madre. Por lo tanto, es importante que el volumen no sobresalga de su volumen madre ni se superponga con otros volúmenes. Si este requisito no se cumple, la simulación probablemente arrojará resultados incorrectos, haciendo que la simulación simplemente ignore los volúmenes afectados. Por lo tanto, si un volumen tiene que sobresalir de su volumen madre como pueden ser, una fibra WLS de un material de centelleo o un volumen adherido a otro por medio de una superficie óptica, entonces, este volumen debe ser dividido y distribuido a diferentes volúmenes madre. Como a menudo no es fácil de detectar dichas superposiciones a simple vista, por ejemplo para geometrías complejas con muchos volúmenes o para pequeñas intersecciones entre volúmenes, para este tipo de situaciones GEANT4 nos proporciona funciones para buscar superposiciones automáticamente[34].
En el último paso, que seria indispensable a la hora de implementar un detector de partículas, se pueden asignar superficies ópticas con ayuda de la clase $G4OpticalSurfaces$ o detectores sensibles a los volúmenes, para simular varias condiciones de superficie diferentes o para especificar acciones definidas por el usuario que se realizarán si una partícula alcanza un cierto volumen, en el ámbito de esta tesis se utilizaran ambas condiciones ya que se requiere tanto detectores sensibles como superficies ópticas que los unan.

6.2.2. Superficies Ópticas

Una superficie óptica esta constituida obligatoriamente por 2 volúmenes independientes conformando una frontera o un limite entre ambas sin dejar entre ellas ningún espacio o segmento vació, al ser un limite que une dos volúmenes figura 6.2, la superficie entre Volume1 y Volume2 no es la misma superficie que la que existe entre Volume2 y Volume1; la definición de superficie es direccional. Cuando hay transporte óptico en ambas direcciones, se deben crear dos superficies. Para esto existen propiedades que, en la mayoría de los casos, nos ayudan a estudiar las distintas formas del paso de partículas a través de varios materiales. Las propiedades de las superficies ópticas se pueden utilizar para simular varias condiciones superficiales diferentes[34]. Por defecto GEANT4 nos proporciona los procesos de superficie óptica de reflexión y refracción, que son, como sabemos, determinados exclusivamente a

partir de los índices de refracción de los dos materiales que forman la superficie (ecuación de Fresnel, ley de Snell). Esto corresponde a la simulación de una superficie perfectamente lisa entre dos dieléctricos. Si es necesario simular otras configuraciones, $G4OpticalSurfaces$ y sus propiedades deben ser definidas por el usuario. Por lo tanto, se pueden usar varios modelos de superficie, tipos de superficie y acabados de superficie.

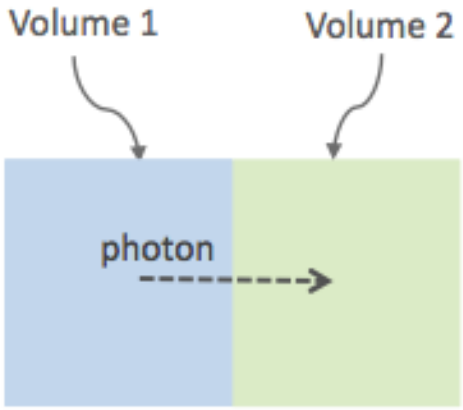

Figura 6.2: Gráfico de un fotón que viaja a través de la superficie entre los dos volúmenes, Volumen1 y Volumen2[34]

Existen dos modelos de simulación para superficies ópticas que se utilizan en estos tipos de límites, estos están disponibles para su uso y pueden ser modificados si se desea. El modelo LUT Davis que fue recientemente implementado al marco de GEANT4 y el modelo UNIFIED tradicional que es un modelo que ya se venia trabajando desde hace varios años.
En el modelo LUT Davis es un modelo para transporte óptico. El modelo se basa en datos de superficie medidos y permite al usuario elegir entre una lista de acabados superficiales disponibles. Se proporciona una superficie rugosa y pulida que se puede usar sin reflector, o en combinación con un reflector especular (por ejemplo el ESR) o un reflector lambertiano (por ejemplo el teflón). En la tabla 6.1 vemos el nombre de las superficies LUTs disponibles, el usuario puede ampliar esta lista de acabados con datos de superficie medidos y personalizados, En la base de datos LUT, se proporcionan parámetros de rugosidad típicos obtenidos de las mediciones para caracterizar el tipo de superficie modelada:

MATERIAL	DESNUDO	TEFLÓN	ESR AIRE	GRASA ESR
PULIDO	Polished_ LUT	PolishedTeflon_LUT	PolishedESR_LUT	PolishedESRGrease_LUT
ASPERO	Rough_LUT	RoughTeflon_LUT	RoughESR_LUT	RoughESRGrease_LUT

Tabla 6.1: Modelo de superficie óptica LUT Davis

- ÁSPERO $Ra = 0,48\mu m$, $\sigma = 0,57\mu m$, $Rpv = 3,12\mu m$
- PULIDO $Ra = 20,8nm$, $\sigma = 26,2nm$, $Rpv = 34,7nm$

en donde

Ra = rugosidad media

σ = rugosidad rms

Rpv = relación pico-valle

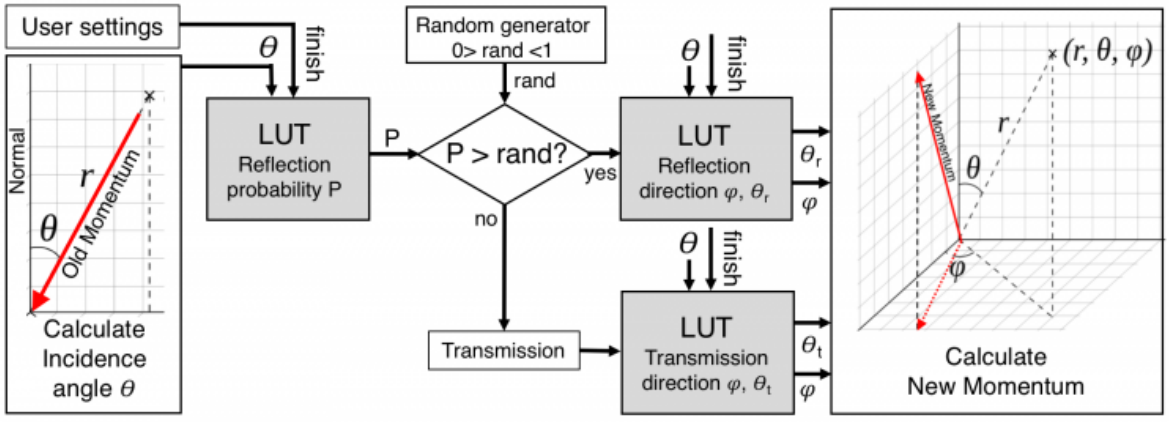

Figura 6.3: Diagrama de flujo del modelo LUT[34]

En la figura 6.3 encontramos el diagrama de flujo para una superficie óptica del modelo LUT, se muestra desde un momento antiguo hasta un momento nuevo, el antiguo momento es el vector unitario que describe el fotón incidente. El fotón reflejado/transmitido es el nuevo momento descrito por dos ángulos φ, θ.

El modelo UNIFIED es el modelo de superficie estándar proporcionado por Geant4 el cual es un modelo unificado para física óptica, este modelo es muy flexible debido a varias posibles combinaciones de varios tipos de superficie, acabados superficiales y variables, por ejemplo, es posible simular superficies pintadas o envueltas con o sin espacios de aire, sin tener que crear volúmenes adicionales para el material de envoltura. Incluso es posible simular una superficie entre un dieléctrico y un metal (con el índice de refracción complejo que es necesario para las ecuaciones de Fresnel). Además, se pueden especificar más propiedades para la superficie, por ejemplo, el espectro de reflectividad de la superficie (sin pasar por las ecuaciones de Fresnel), su rugosidad y los modelos de reflexión que se aplicarán a una determinada superficie. El modelo UNIFIED permite al usuario controlar la intensidad radiante de la superficie, en la figura 6.4 se observa: Lóbulo especular, espiga especular, espiga de retrodispersión (mejorada en superficies muy rugosas) y reflectividad (distribución lambertiana o difusa). La suma de las cuatro constantes está restringida a la unidad. En ese modelo, los vectores normales micro-facetados siguen una distribución gaussiana definida por sigmaalpha (σ_α) dada en grados.

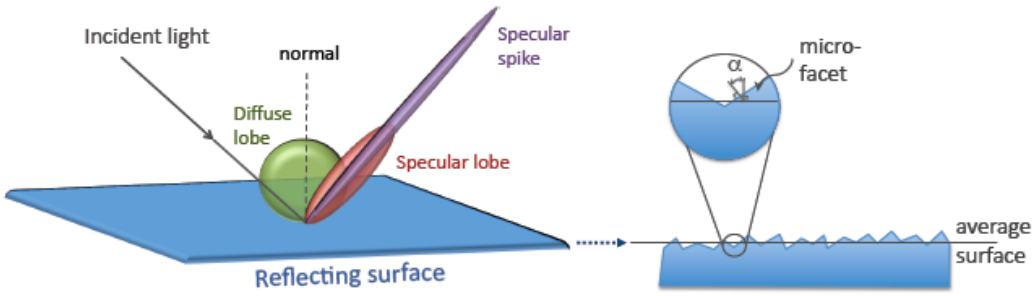

Figura 6.4: Tipos de reflexión y microfacetas[34]

Ahora bien, en caso de que un fotón se refleje, son posibles cuatro tipos de reflexión. Las probabilidades de los tres primeros vienen dadas por los siguientes tres vectores de propiedad ya establecidos dentro de GEANT4[34]:

- SPECULARSPIKECONSTANT da la probabilidad de reflexión especular sobre la superficie media normal.

- SPECULARLOBECONSTANT da la probabilidad de reflexión especular sobre la superficie normal de la micro faceta.
- BACKSCATTERCONSTANT da la probabilidad de reflexión en la dirección de la que proviene el fotón óptico.
- La reflexión LAMBERTIANA (difusa) ocurre cuando no ocurre ninguno de los otros tres tipos de reflexión. La probabilidad de reflexión lambertiana viene dada por uno menos la suma de las otras tres constantes.

Cuando un fotón se refracta, el ángulo de refracción se calcula a partir de la superficie normal (de la superficie media para pulido y de la micro faceta para rugoso) y los índices de refracción de los dos medios. Cuando un fotón óptico alcanza una capa pintada (superficie óptica), la probabilidad de reflexión viene dada por la propiedad del vector REFLECTIVIDAD. En caso de que la pintura esté en el interior de la superficie, se ignoran los índices de refracción de los medios, y cuando el fotón se refleja, sufre una reflexión lambertiana. Cuando la pintura (superficie óptica) está en el exterior de la superficie, primero se calcula si el fotón se refleja en la interfaz entre los dos medios, utilizando el método descrito en la sección anterior. Sin embargo, en este caso se utiliza el índice de refracción dado por la propiedad vectorial RINDEX de la superficie. Cuando el fotón se refracta, se refleja utilizando la reflexión lambertiana con una probabilidad de REFLECTIVIDAD. Luego, de nuevo, tiene que pasar el límite entre los dos medios. Para esto, el método descrito en la sección anterior se utiliza una y otra vez, hasta que el fotón finalmente se refleja de nuevo en el primer medio o es absorbido por la pintura. Una superficie dieléctrica puede tener una propiedad de transmitancia dependiente de la longitud de onda. Si esto se especifica para una superficie, sobrescribe la probabilidad de la ley de Snell. Esto permite la simulación de recubrimientos antirreflectantes.

6.2.3. Partículas Primarias

La clase $G4ParticleGun$ es la responsable de generar un evento de la clase $G4Events$ cuando ejecutamos (corremos) una simulación con $G4Run$ en GEANT4. Todas estas clases en

conjunto son las responsables de establecer la inicialización de los eventos que se ejecutan dentro de la simulación. En el ultimo paso $G4Run$ es la responsable de la recolección de los eventos y con quien comparte las mismas condiciones que el detector, sin esto el usuario no puede cargar la geometría o los demás procesos físicos implementados al detector. Aquí se deben especificar tanto el número de las partículas primarias como sus respectivas propiedades implicadas en cada ejecución del programa[34]. Propiedades como el tipo de partícula, su posición inicial, su energía cinética o momento, su dirección de trayectoria, entre otras, pueden ser proporcionados dentro de la simulación ya sea con valores fijos o como distribuciones de probabilidad. Para el ámbito de la presente tesis haremos énfasis en la implementación de los denominados fotones ópticos, que son un tipo de partícula dentro del marco de simulación de GEANT4 especializada en el estudio del transporte óptico y sus correspondientes propiedades involucradas.

Como el fotón óptico se utiliza para la luz cerenkov y la luz de centelleo, es esencial utilizar esta partícula en el estudio que se esta realizando, sin embargo, antes de discutir cómo se va a usar el seguimiento óptico de fotones, debe mencionarse que hay algunas desventajas en el uso del transporte óptico. Primero, el tiempo de simulación aumentará dramáticamente. Por ejemplo, la mayoría de los centelladores utilizados en simulaciones generan del orden de 10.000 fotones ópticos a 511 keV, lo que significa que aproximadamente 10.000 partículas más tienen que ser rastreadas por cada fotón de aniquilación que se detecta. Aunque el seguimiento de fotones ópticos es relativamente rápido, una simulación con seguimiento óptico de fotones puede ser fácilmente un factor mil más lento que uno sin él. Finalmente, para realizar simulaciones ópticas, se necesitan muchos parámetros para los materiales y superficies, algunos de los cuales pueden ser difíciles de determinar.

Un fotón óptico con una longitud de onda de 530 nm corresponde a un fotón óptico de energía igual a 2.34 eV (aproximación: 1240/E(eV) = longitud de onda (nm)). Si un fotón óptico no se le asigna una polarización en producción, este no puede ser disperso por Rayleigh y por lo tanto esta propiedad en especifico es ignorada al momento de la recolección de los eventos dentro del detector.

6.2.4. Acciones definidas por usuario

Además de las clases obligatorias de GEANT4, que el usuario debe introducir para definir las propiedades necesarias de la simulación, existen otras clases que se pueden aplicar para acceder a la simulación, estas son las $classes$ $G4Run$, $G4Event$, $G4Track$ o $G4Step$. De este modo, el usuario tiene la posibilidad de recopilar los datos de simulación deseados y/o especificar acciones definidas por el mismo. Por ejemplo, las partículas se pueden "matar" (es decir, detener la simulación de su trayectoria) cuando alcanzan volúmenes específicos o un número de pasos determinados, se puede determinar el número de Cherenkov o fotones de centelleo por ejecución/evento/partícula primaria, etc.

En caso de que dichas acciones definidas por el usuario se ejecuten para partículas que alcancen volúmenes específicos, esto se puede simplificar utilizando el concepto de detectores sensibles (Sensitive Detector) en GEANT4. Cada volumen se puede declarar como $G4SensitiveDetector$. Como resultado, la acción definida por el usuario que se ha asignado a un determinado $G4SensitiveDetector$ se ejecuta si una partícula alcanza el volumen correspondiente. El uso de detectores sensibles evita la necesidad de una verificación definida por el usuario del volumen actual para cada $G4Step$ (evento) y simplifica la definición de diferentes acciones para diferentes volúmenes. Además, el detector sensible puede crear un $G4Hit$ para cada partícula que alcanza el volumen. Estos $G4Hit's$ contienen datos definidos por el usuario de las partículas (por ejemplo, tipo de partícula, cuadri-momentos, energía depositada, etc.) y se pueden recopilar durante un evento. Al final del evento, los $G4Hit's$ se pueden digitalizar aplicando criterios definidos por el usuario. Por lo tanto, esto permite la posibilidad de la simulación de una respuesta de detector realista.

Capítulo 7

Simulación del Detector y Resultados

A continuación se presenta el diseño estructurado del detector de centelleo, el cual consta de una placa centelladora plástica acoplada a un dispositivo detector de luz, en nuestro caso fotomultiplicadores de silicio (SiPM). La simulación detalla la geometría del detector, las propiedades físicas tanto de los materiales que lo componen así como también de las partículas generadas durante la interacción, y finalmente muestra el rastreo explicito de fotones creados en el material centellador que permite registrar datos y estudiar la respuesta del mismo.

7.1. Geometría del detector

El diseño del detector esta implementado en la clase $DetectorConstruction$, que es la responsable de toda la geometría del detector dentro de la aplicación construida para simular la interacción radiación-materia usando GEANT4. En esta clase también se incluye el tipo de material involucrado y las propiedades ópticas que serán características del mismo. Nuestra geometría consiste de una placa de material centellador plástico con dimensiones 24cm $\times$ 24cm $\times$ 1cm, en las figuras 7.1, 7.2 y 7.3 se puede observar la placa desde diferentes puntos de vista. Sus propiedades como la reflectividad, rendimiento de centelleo, atenuación, emisión, etc. son implementadas dentro de la clase $G4MaterialPropertiesTable$, que nos proporciona una lista de propiedades que serán únicas del material de nuestro detector. La tabla 7.1 resume todas las características que posee el material centellador de referencia BC408, el más usado para este tipo de experimentos.

La geometría de la placa a su vez esta rodeada por un revestimiento de pintura reflectante con referencia EJ-510, que es frecuentemente usada para centelladores de plástico. Esta pintura es de color blanco brillante y consiste de un pigmento de dióxido de titanio y una base de

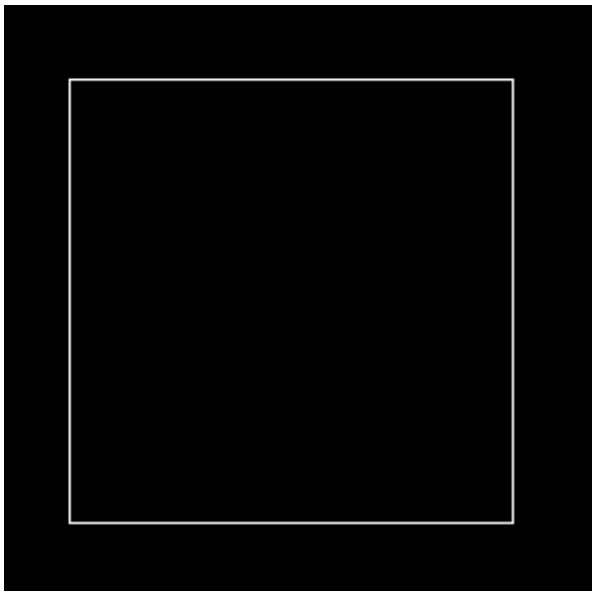

Figura 7.1: Vista frontal de la placa plástica centelladora.

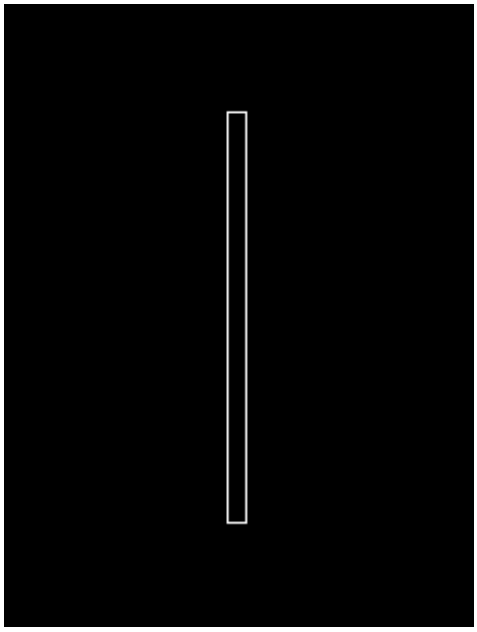

Figura 7.2: Vista lateral de la placa plástica centelladora.

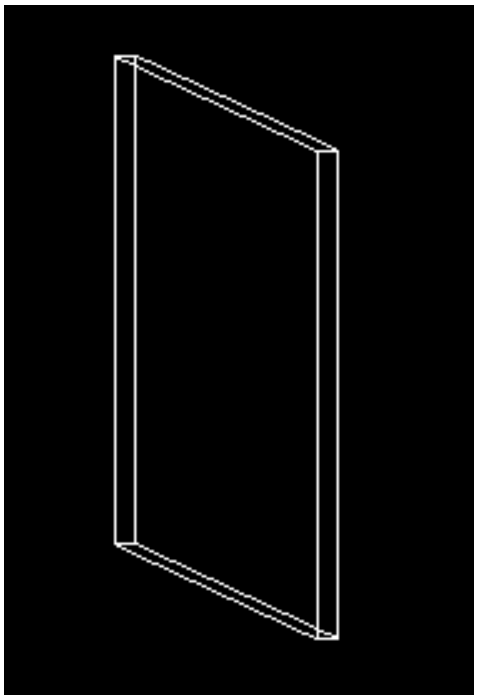

Figura 7.3: Vista completa de la placa plástica centelladora.

Centellador BC408	
Base	poliviniltolueno
Radio atómico (H:C)	11:10
Densidad (g cm^{-3})	1.032
Índice de refracción	1.58
Salida de luz (%)	64
Rendimiento de centelleo (fotones/MeV)	11136
Pico de emisión (nm)	425
Longitud de atenuación (cm)	210
Rendimiento lento/rápido	0.27
Tiempo de decaimiento rápido (ns)	2.1
Tiempo de decaimiento lento (ns)	14.2

Tabla 7.1: Propiedades del material centellador plástico utilizado para la simulación de la placa.

pintura soluble en agua seleccionada especialmente para tener una excelente resistencia al amarilleo así como también una buena adherencia, sus propiedades se muestran en la tabla 7.2. Para simular este revestimiento dentro de nuestra aplicación de GEANT4, lo que haremos es colocar una superficie óptica que presente las mismas características que la pintura real, este revestimiento reflectante se trata como una superficie óptica de metal dieléctrico que se asume como si estuviera pulida, dicha superficie ira en dirección desde el volumen madre hacia el interior de la placa de centelleo, como se vera más adelante, nos servirá para ensamblar de manera adecuada y ópticamente correcto el SiPM con la placa.

Propiedades	EJ-510
Espesor típico de recubrimiento de 3 capas (mm)	0.11
Densidad típica de la capa seca (mg/cm^2)	13
N°. de Ti átomos por cm^2 ($\times 10^{19}$)	6.71
N°. de C átomos por cm^2 ($\times 10^{20}$)	1.12
N°. de átomos de H por cm^2 ($\times 10^{20}$)	2.25
N°. de átomos de O por cm^2 ($\times 10^{20}$)	1.90

Tabla 7.2: Propiedades de la pintura reflectante EJ-510 aplicada en centelladores plásticos

Para la lectura de los fotones generados en la placa de centelleo se considera el uso de fotomultiplicadores, estos además de realizar la lectura de los fotones convierten este conteo en una señal electrónica para luego ser digitalizada mediante algunas tarjetas electrónicas de procesamiento. Para llevar a cabo una simulación en un escenario realista, se implementa un fotomultiplicador de silicio o SiPM, en el cuál los fotones depositarán energía, está energía se convertirá en la señal electrónica mediante algún tratamiento numerico, pero esto esta fuera de los objetivos del presente trabajo. El SiPM que se utiliza en la simulación es de la marca Onsemi referencia MicroC-Series 60035 cuyas características se muestran en la tabla 7.3.

Para la prueba de simulación en GEANT4 se escribió una clase aparte en la aplicación que contiene el SiPM como un detector sensible (SD por sus siglas en ingles), esto está unido ópticamente a la placa de centelleo, como se puede ver en la figura 7.4(a). El fotocátodo

Características Generales	SiPM 60035
Tamaño (mm)	6
Área activa (mm^2)	6×6
No. de microcélulas	18980
Factor de llenado de microcélulas (%)	64

Tabla 7.3: Características generales de un Fotomultiplicador de Silicio (SiPM) marca Onsemi referencia MicroC-Series 60035.

del SiPM se trata como un volumen sensible y la interfaz cátodo-centellador se trata como una superficie óptica de metal dieléctrico, configurada como pulida y modelo $UNIFIED$. También se consideró que la reflectividad sea cero y una eficiencia de $20\,\%$ para la detección de fotones.

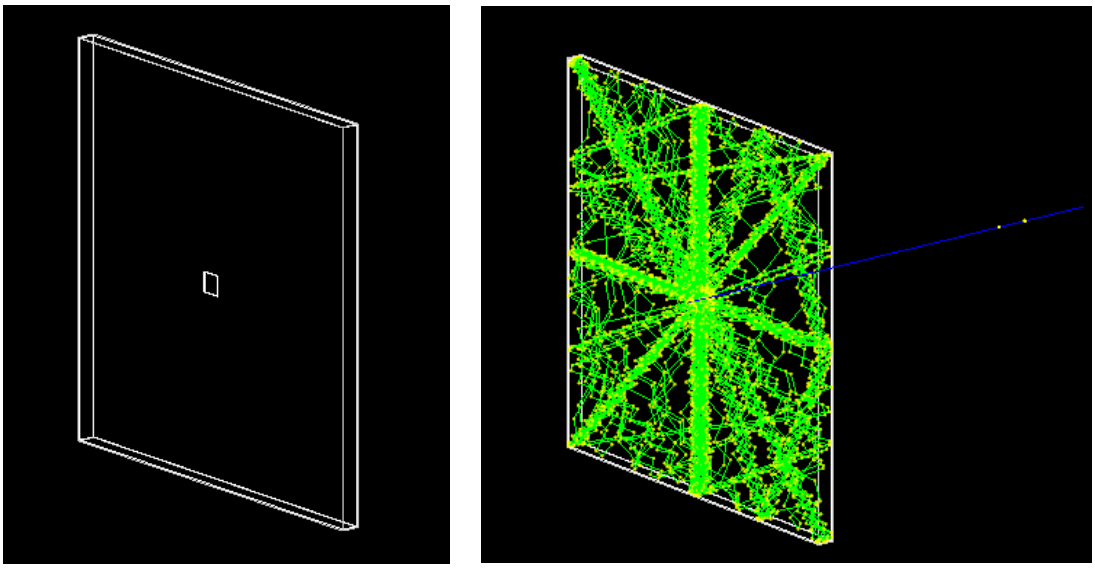

Figura 7.4: Placa plástica centelladora y el SiPM (cuadrado pequeño) unidos ópticamente. En la figura izquierda se muestra el detector sin eventos simulados y en la figura derecha se muestra el detector con la trayectoria de un muón (azúl) disparado con una energía de 1 GeV. Las trayectorías verdes son los fotones ópticos generados en el material centellador.

7.2. Física y procesos ópticos

Todos los procesos físicos y ópticos relevantes para el desarrollo de la simulación, así como todas las partículas involucradas y sus respectivas componentes, se implementan en GEANT4 dentro de la clase $PhysicsList$. Se inicia con los primeros procesos relevantes que son los electromagnéticos como es el caso de la ionización, bremsstrahlung, dispersión múltiple, producción de pares, dispersión Compton y efecto fotoeléctrico. Dentro del material centellador que compone al detector, los procesos ópticos deben incluir la generación de fotones de centelleo, la emisión Cherenkov, la absorción masiva, la dispersión de Rayleigh y los procesos de contorno (como lo son la reflexión, refracción y absorción de luz).
El proceso dominante de generación de fotones en el material BC408 es el proceso de centelleo. El número de fotones de centelleo se genera según una distribución gaussiana que tiene como valor medio:

$$N_{ph}^{scint} = Y_{scint} \times E_{dep}, \tag{7.57}$$

donde Y_{scint} y E_{dep} son la producción de fotones de centelleo del material y la energía total depositada en el detector, respectivamente. La desviación estándar de esta distribución está dada por $\sigma = \sqrt{N_{ph}^{scint}}$. Si N_{ph}^{scint} es menor a 10, los fotones emitidos obedecen una distribución de Poisson con valor medio N_{ph}^{scint}.
Si se considera que los fotones ópticos son emitidos isotrópicamente a lo largo del área del material del detector, podemos conocer la distribución angular de los fotones. En la aplicación de la simulación construida en GEANT4 es posible obtener una medida de los eventos correspondientes a los hits marcados por el material SD (detector sensible); con ayuda de la clase $G4AnalysisManager$ podemos extraer la información que nos proporciona la simulación, dicha información puede ser almacenada para luego realizar un análisis correspondiente. En nuestro caso hacemos uso del software ROOT proporcionado por CERN, el cuál es una herramienta computacional de almacenamiento y análisis de datos de código abierto (creado bajo el lenguaje C++) utilizado en la física de altas energías. En este orden de ideas, se opta por mostrar los resultados utilizando histogramas de frecuencia, que son ideales a la

hora de estudiar el comportamiento estadístico de un número apreciable de datos dentro de un mismo sistema físico y en nuestro caso resulta idóneo puesto que se trabaja con una gran cantidad de datos registrados.

En la figura 7.5 se muestra la distribución angular de los fotones de centelleo, estos son producidos por un flujo muones verticales de energía de 1 GeV que cruzan la placa en su centro. Los datos de la figura 7.5 muestran, como se espera, una distribución angular isotrópica. La correspondiente distribución angular para los fotones de Cherenkov alcanza su valor máximo en $\theta = 50{,}7^{\circ}$ el cuál es el ángulo de Cherenkov esperado que recordemos es calculado mediante la relación:

$$\cos\theta_c = \frac{1}{n\beta} \tag{7.58}$$

donde n es el índice de refracción del material y β es la fracción v/c con v siendo la velocidad de la partícula (fotones ópticos).

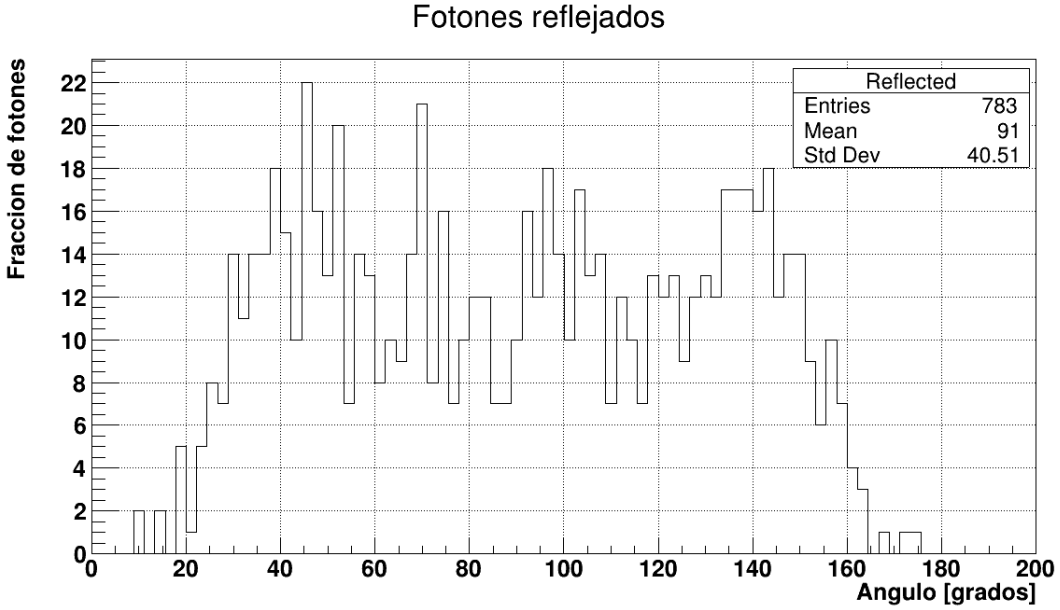

Figura 7.5: Distribución angular de fotones de centelleo generados por la interacción de un flujo de muones de energía 1 GeV con la placa. En este caso los fotones poseen una polarización arbitraria en la superficie óptica.

En la figura 7.6 se muestra la distribución temporal de los fotones de centelleo, producidos por un flujo de muones verticales de energía de 1 GeV que cruzan la placa en su centro. Los datos del tiempo de emisión en la figura 7.6 se pueden ajustar muy bien con una función exponencial de la forma:

$$c_{fast} \exp\left(-\frac{t}{\tau_{fast}}\right). \tag{7.59}$$

Este ajuste produce exactamente la constante de decaimiento rápido asociada al material del centellador ($\tau_{fast} = 1/0{,}4775 = 2{,}1$ ns) mientras que la emisión de Cherenkov es instantánea.

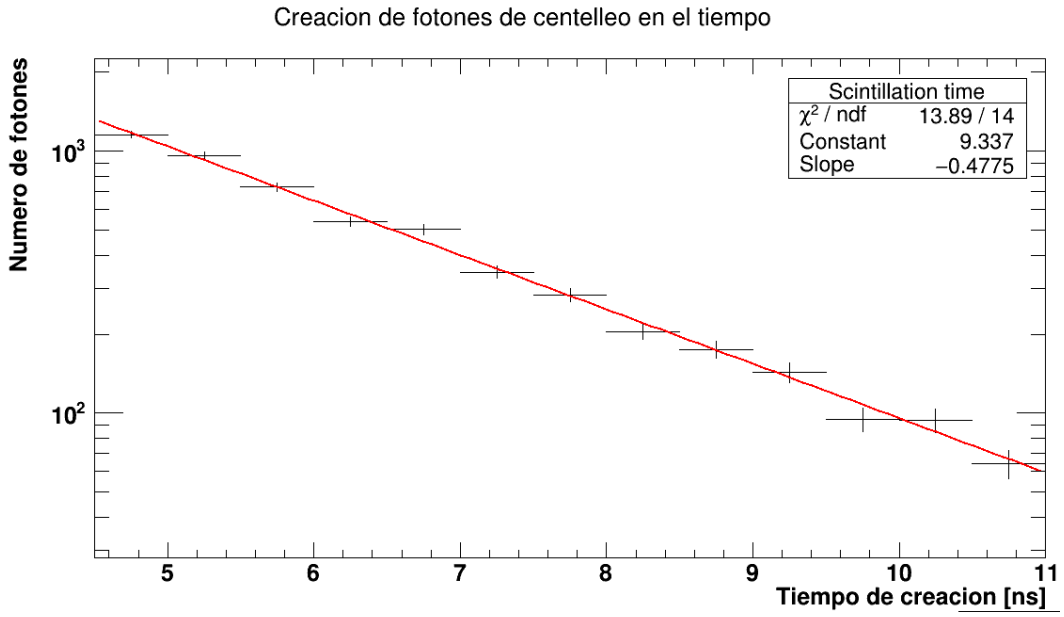

Figura 7.6: Distribución del tiempo de emisión de fotones ópticos de centelleo generado por muones de 1GeV.

7.2.1. Energía depositada en el centellador

Vamos ahora a analizar la fracción de energía depositada en el centellador después de atravesar el revestimiento reflectante (superficie óptica de metal dieléctrico), en la figura 7.7 podemos ver la distribución de energía depositada en el detector (centellador plástico) por

muones de energía 1 GeV disparados a 5 cm de su superficie, esto normalmente se hace con el fin de poder realizar una comparación entre la señal arrojada por la simulación y el laboratorio, pero esto va más allá de los objetivos de este trabajo. De la figura se puede ver que el valor medio de energía depositada por los muones es de 3.13 MeV.

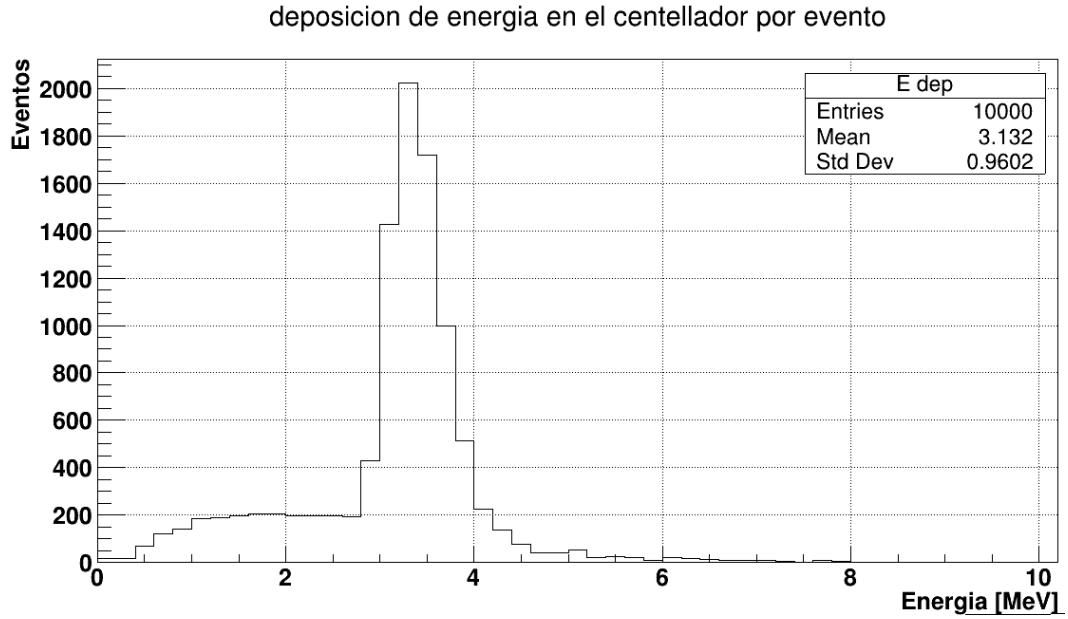

Figura 7.7: Energía depositada en el centellador por muones de energía de 1GeV por evento.

Gracias a la formula de Bethe-Bloch (4.19) sabemos que los muones de baja energía depositan en el centellador plástico toda su energía restante después de atravesar el recubrimiento del detector, pero la energía depositada disminuye para los muones más energéticos ya que la probabilidad de escape es mucho mayor comparada con los muones menos energéticos.
La energía depositada en el centellador se debe principalmente a procesos de pérdida de energía como ionización o bremsstrahlung y la fracción transferida al centelleo se modela en la simulación mediante la producción de fotones de centelleo en el plástico. El resultado está de acuerdo con el hecho conocido de que el rendimiento típico de luz (fracción de energía depositada que se transfiere a la luz cuando el evento primario deposita toda su energía en el plástico) en centelladores orgánicos es de alrededor de 2 a 4 $\%$, considerando que parte de la energía en nuestro caso se libera en el recubrimiento y el volumen limitado del plástico.

7.2.2. Fotones de centelleo

La colección de luz por parte del centellador ocurre debido a dos procesos básicos: uno de ellos es el escape a través de los limites del centellador y la otra forma es a través de la absorción por parte del material centelleante. Para detectores pequeños, como es en nuestro caso, el ultimo efecto es casi despreciable, ver figura 7.8. Solamente cuando las dimensiones del detector son tales que la longitud total recorrida por los fotones es comparable con la longitud de atenuación, en ese caso la absorción jugara un rol importante dentro de la colección de luz en el detector. Este parámetro se define como la longitud después de la cual la intensidad de luz es reducida un factor de e^{-1}, la intensidad de la luz como una función de la longitud, esta dada por:

$$L(x) = L_0 \exp\left(\frac{-x}{l}\right), \tag{7.60}$$

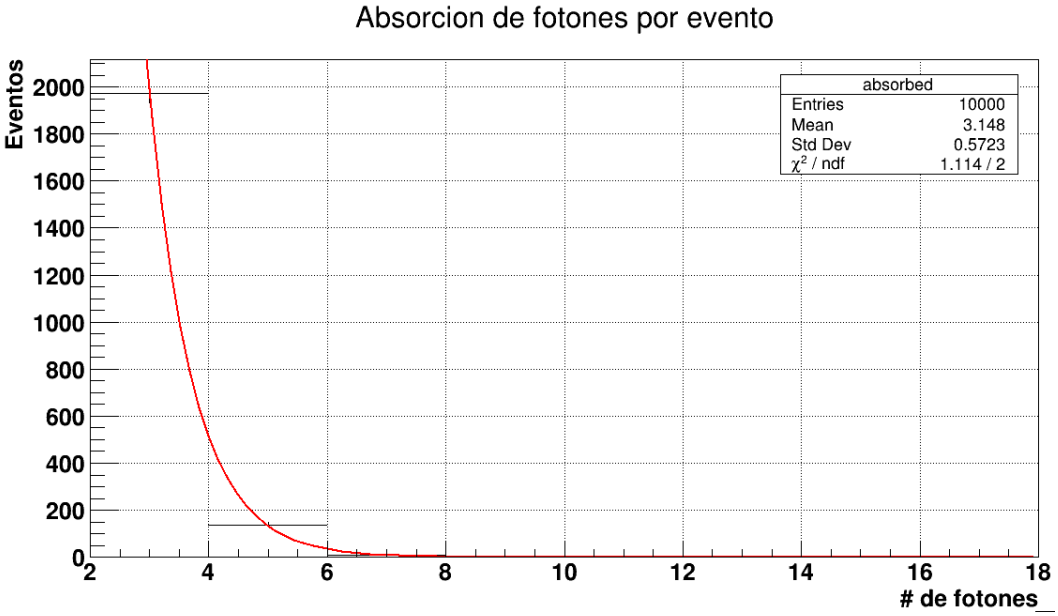

Figura 7.8: Absorción de fotones ópticos dentro del material centellador. La linea roja se ajusta muy bien a una función exponencial negativa de la forma $e^{10}e^{-x} = L_0e^{-x}$ con L_0 siendo la intensidad inicial de los fotones.

donde l es la longitud de atenuación, x es la longitud recorrida por la luz y L_0 la intensidad

de luz inicial. De donde la longitud de atenuación típica es del orden de 1 m o más, eso claro nos muestra que solo los detectores más grandes son afectados.

Ahora bien, el proceso más importante para la colección de luz en el centellador es debido a la absorción en los limites del material, como se muestra en la figura 7.9, en donde se ve un aumento significativo en comparación con el anterior proceso, puesto que este si tiene consideración directa independientemente del tamaño del detector.

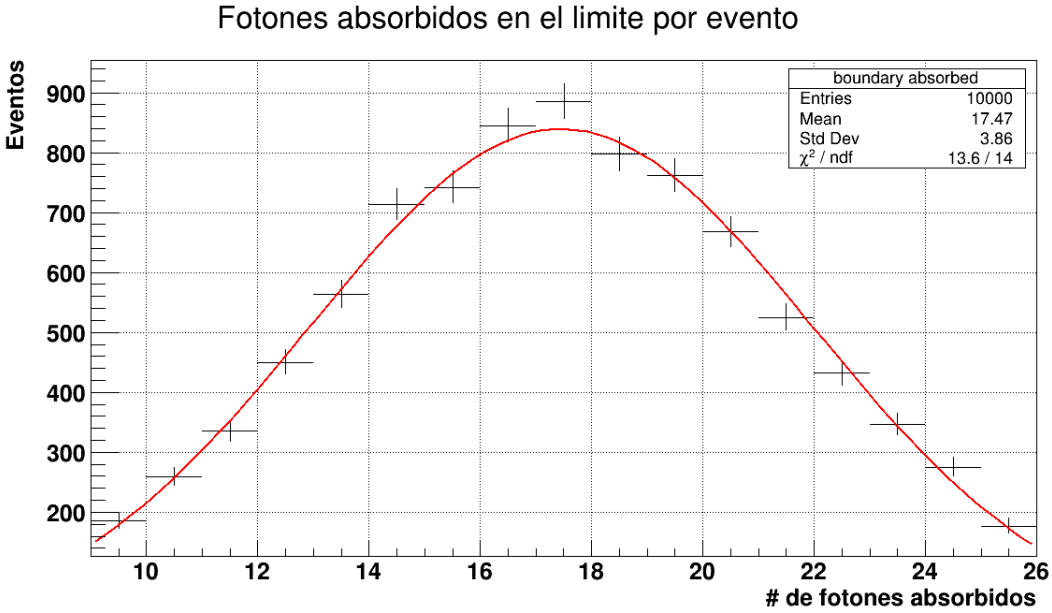

Figura 7.9: Absorción de fotones ópticos en los limites del material centellador. La linea roja se ajusta a una distribución gaussiana de la forma $Ae^{-0,5(\frac{x-Mean}{\sigma})^2}$

Esta absorción, por su supuesto, esta directamente relacionada con la eficiencia y la resolución energética que tendrá nuestro detector. En la práctica se sabe que la dependencia con el punto de emisión de luz y el volumen del detector pueden ser despreciables, ya que el tipo de detectores usados en laboratorios de física nuclear son por lo general pequeños ya que los detectores con geometrías más complejas y volúmenes mucho más grandes pueden suponer algunas complicaciones a la hora de ponerlos en funcionamiento y para incrementar su eficiencia de colección luz de centelleo se deben emplear métodos complejos para su aplicación.

7.2.3. Fotones Cherenkov

El número total de fotones de Cherenkov, así como la distribución de energía o longitud de onda de los fotones de Cherenkov, depende del valor β de la partícula que atraviesa el material y del espectro del índice de refracción del material. Para validar la correcta simulación de la radiación de Cherenkov en GEANT4, muones con $\beta = 1$ han sido simulados atravesando la placa de centelleo. El número de fotones Cherenkov por evento simulado se ha registrado en el histograma mostrado en la figura 7.10, se puede ver que el valor medio de fotones Cherenkov producidos por evento es de 27.34.

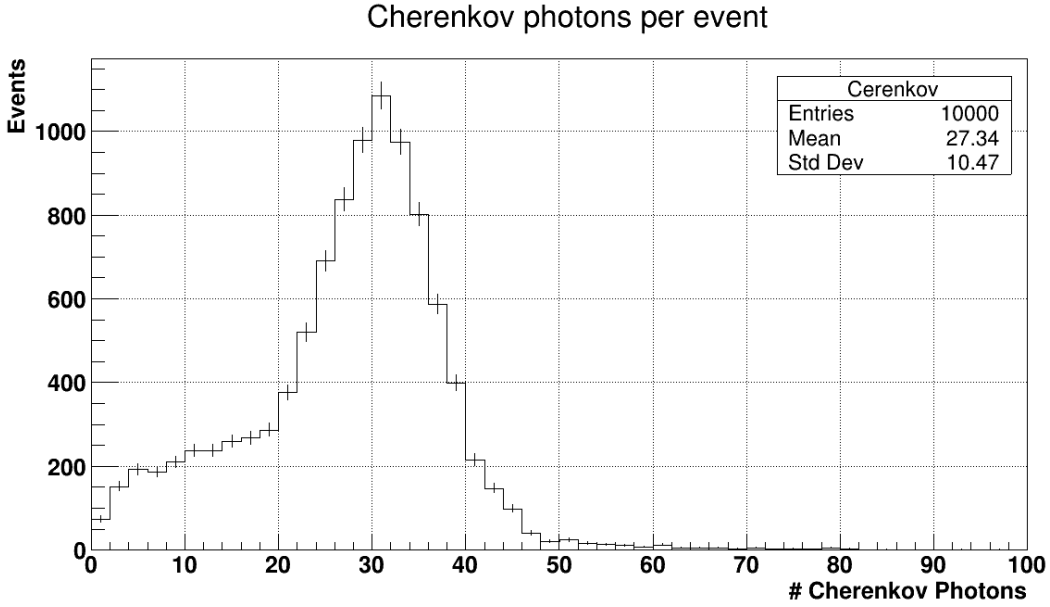

Figura 7.10: Distribución del número de fotones Cherenkov por evento dentro de la placa plástica de centelleo producidos por el paso de un flujo muones con energía de 1 GeV a 5 cm del detector.

Como bien sabemos los fotones de Cherenkov se simulan en GEANT4 por defecto [3]. El tiempo y la posición de una emisión de fotones Cherenkov se basan en cantidades conocidas al comienzo del paso de una partícula cargada. Los fotones de Cherenkov se emiten en la dirección del cono de Cherenkov con un ángulo que depende del mismo modo de la energía de la partícula cargada primaria (muón). Por el contrario, los fotones de centelleo se emiten

isotrópicamente en el detector. El espectro de energía se simula siguiendo fórmulas conocidas. En la Figura 7.11 se muestra el espectro de energía simulado de Cherenkov y los fotones de centelleo donde los espectros de energía de los componentes rápido y lento se definen idénticamente.

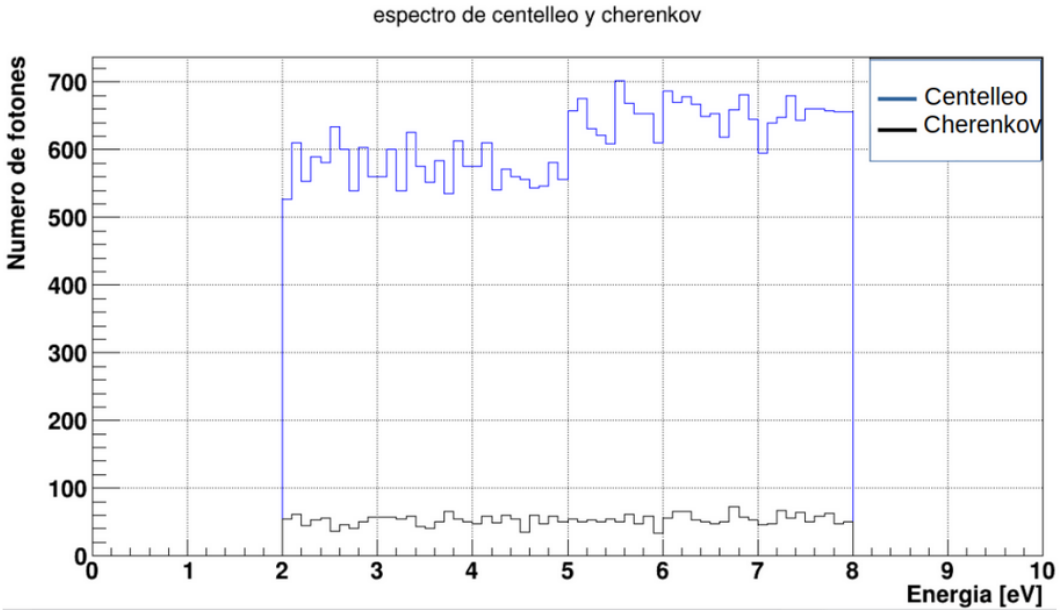

Figura 7.11: Gráfica del espectro de energía de centelleo (linea azul) y el espectro de energía Cherenkov (linea negra)

Podemos entonces concluir que al rededor de un $10\,\%$ de la luz emitida por el material centellador es convertida en radiación de Cherenkov. Cabe resaltar que en la simulación se impuso una media de 10000 eventos primarios con energía de 1 GeV disparados directamente al detector con recubrimiento de material dieléctrico pulido UNIFIED.

7.2.4. Fotones Reflectados

El uso de una capa reflectante es la técnica más común para aumentar la captación de fotones ópticos dentro del material centellador, lo que supone un punto clave en cualquier experimento de este tipo. También sabemos que la fracción de fotones perdidos cuando el detector se encuentra sin este recubrimiento es considerable en comparación con el número total de

fotones producidos a la hora de realizar un conteo que alcance al SiPM, como consecuencia la señal total recogida en el fotodetector se reduce (dependiendo de la energía y el volumen del detector). Por lo tanto, es evidente que es muy recomendable utilizar este recubrimiento de metal dieléctrico pulido.

Las figuras 7.12 (sin capa reflectante) y 7.13 (con capa reflectante) comprueban que si la interfaz del revestimiento del detector plástico está perfectamente pulida, al mismo tiempo se mejora la reflectividad del revestimiento haciendo que los fotones recogidos por el SiPM aumente entre un $30\,\%$ y un $80\,\%$, esto se puede ver en el número de entradas en cada distribución.

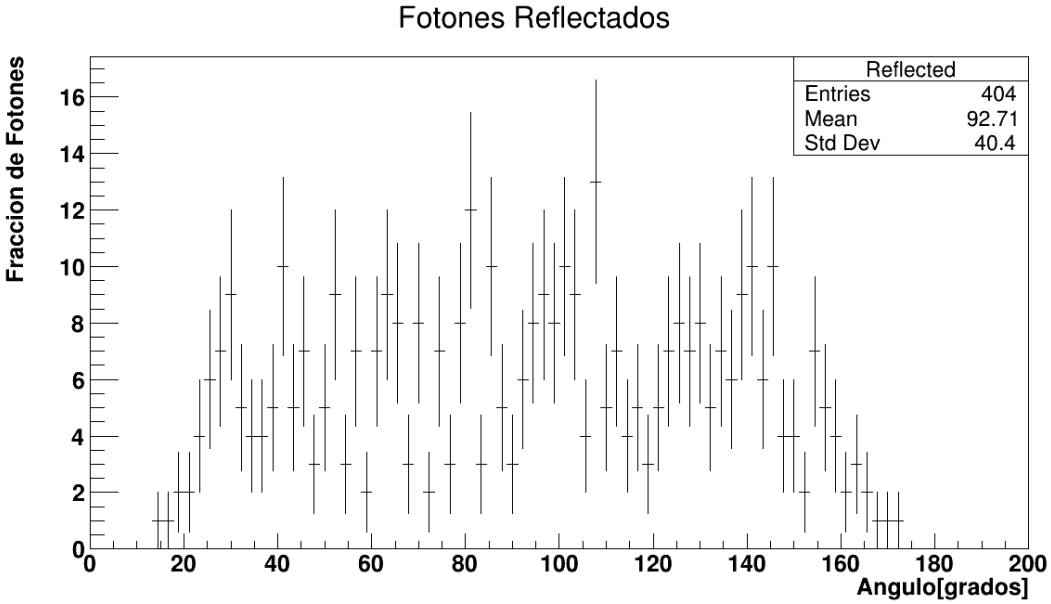

Figura 7.12: Gráfica de la fracción de fotones ópticos transmitidos en función del ángulo de reflexión “sin” superficie óptica de recubrimiento en el detector

Un tema crucial que no ha sido analizado previamente en la literatura pero que se trae a colación es que el espesor del recubrimiento impone un límite inferior a la energía de los eventos primarios que podrían llegar a atravesarlo y, por tanto, que sea capaz de producir una

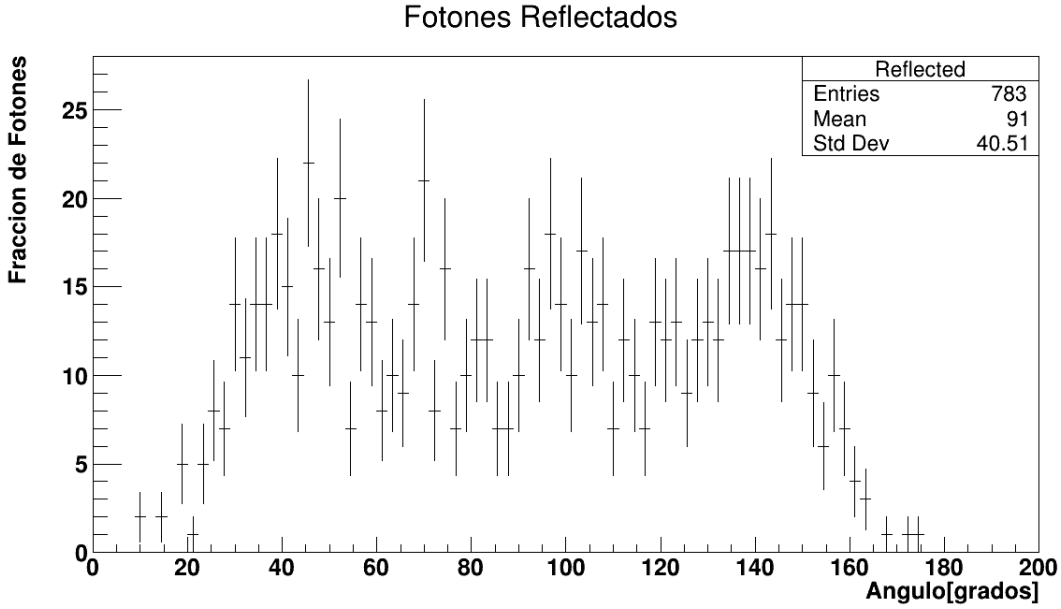

Figura 7.13: Gráfica de la fracción de fotones ópticos transmitidos en función del ángulo de reflexión “ con” superficie óptica de recubrimiento en el detector

señal detectable. Esto es de gran importancia en los experimentos. Por ello se ha determinado que, con el espesor estándar de 0,11 mm, los muones de energías muy altas, así como las otras partículas, pierden energía en el recubrimiento (que no se transfiere al centellador) en una cantidad significativa. Este efecto es menos importante a medida que aumenta la energía primaria de las partículas por lo que dentro de los parámetros que se estableció se puede despreciar este efecto.

La fracción de energía depositada en el plástico aumenta con el volumen pero depende en gran medida del tipo y la energía de los eventos primarios, ya que la probabilidad de escapar del plástico aumenta con la energía primaria y las interacciones en el plástico son muy diferentes para los muones de alta y baja energía. En las figuras 7.14, 7.15 y 7.16 podemos notar este comportamiento para energías de 1, 10 y 100 GeV. Cabe resaltar que esta fracción depende también de la energía perdida en el recubrimiento como se comentó anteriormente.

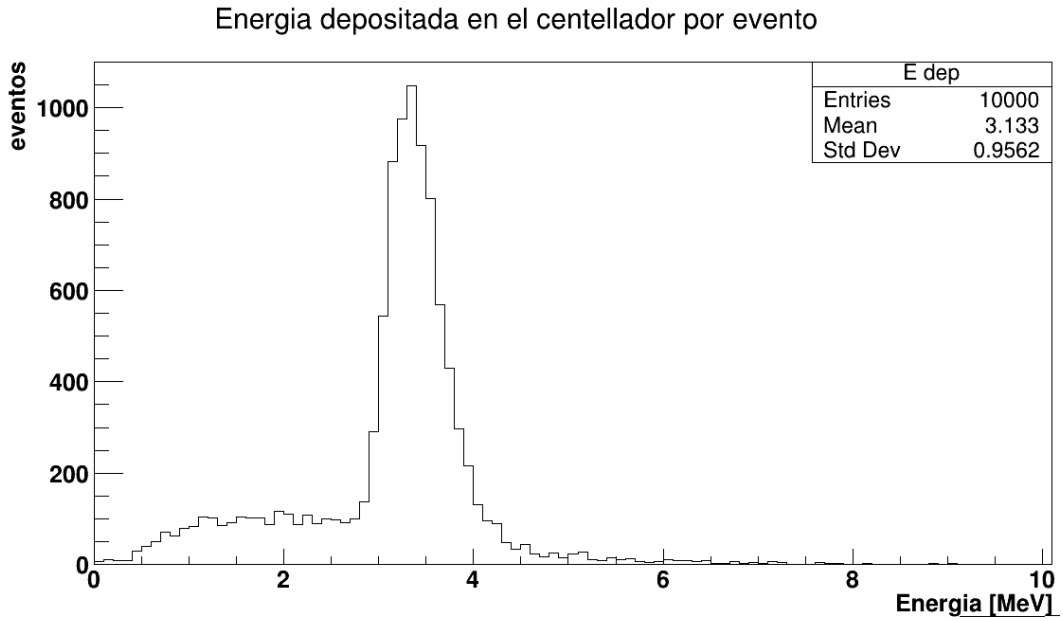

Figura 7.14: Gráfica de energía depositada en el material centellador en función del número de eventos. La energía primaria usada en el proceso es de 1 GeV para cada uno de los 10000 eventos totales

La cantidad de fotones ópticos (en comparación con el número total de fotones producidos figura 7.9) captada directamente por el fotodetector (Hits en el SiPM figura 7.18) es bastante bajo y la mayor parte de los fotones se colectan después de varias reflexiones dentro del material centellador plástico, siendo esta fracción de fotones capturados independiente de la energía de los eventos primarios pero ligada a la cantidad de partículas involucradas en dicho proceso (ver figuras 7.14, 7.15 y 7.16).

El proceso de creación de fotones de centelleo (ver figura 7.6) que se encuentra en el orden de los nanosegundos (ns) en nuestro detector, lo que resulta ser casi instantáneo, el material centellador plástico emite los fotones ópticos producto de las múltiples colisiones entre el material y los muones casi al momento de su interacción, es por ello que la emisión de fotones producto de la propiedad de luminiscencia que posee el centellador no sera considerable a menos que se trabajen con detectores más grandes, comparables al orden de los metros, cuyo recorrido libre medio para las partículas incidentes (muones) sea mucho mayor que la

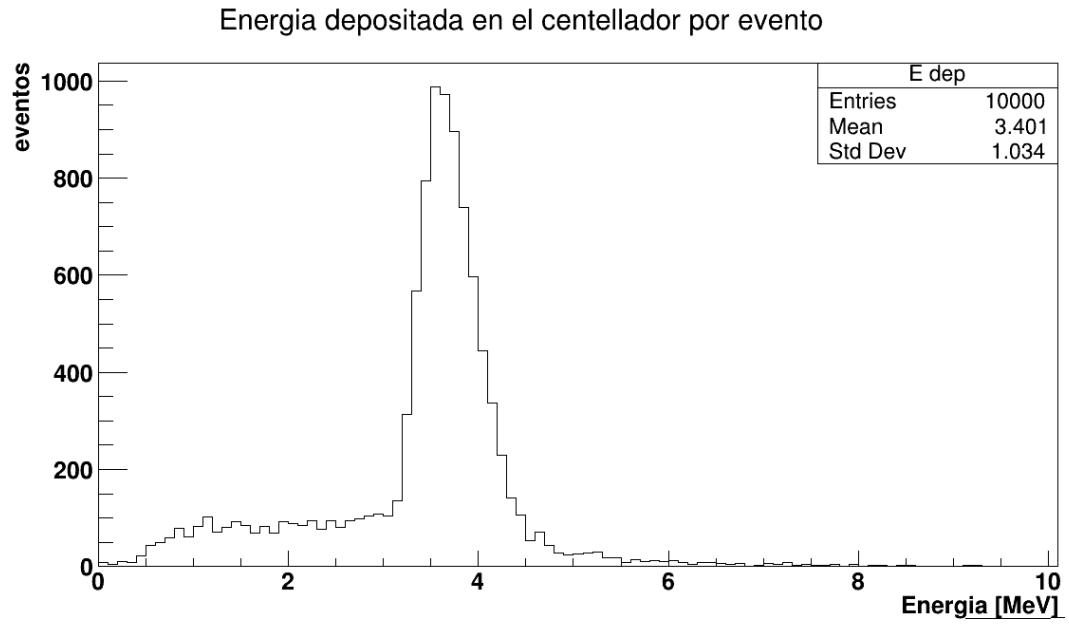

Figura 7.15: Gráfica de energía depositada en el material centellador en función del número de eventos. La energía primaria usada en el proceso es de 10 GeV para cada uno de los 10000 eventos totales

longitud de onda de los fotones emitidos.

7.3. Respuesta del fotomultiplicador de Silicio (SiPM)

El SiPM no puede activarse como un detector sensible normal porque el volumen sensible no permite que los fotones lo atraviesen. Más bien, los detecta en el proceso $OpBoundary$ en función de una eficiencia establecida en la superficie del volumen:

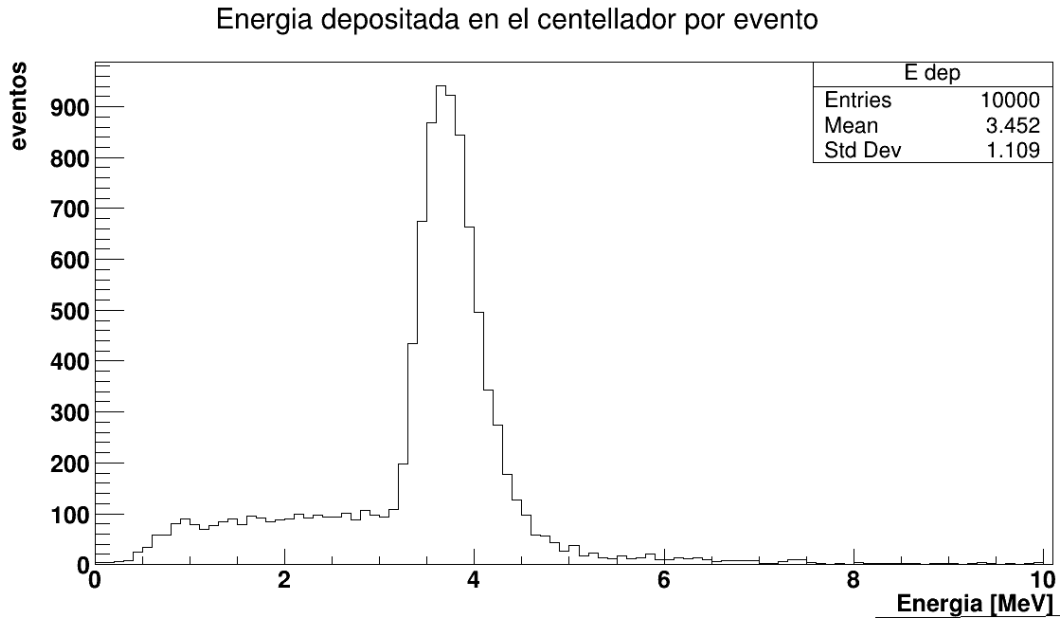

Figura 7.16: Gráfica de energía depositada en el material centellador en función del número de eventos. La energía primaria usada en el proceso es de 100 GeV para cada uno de los 10000 eventos totales

```
G4OpticalSurface* photocath_opsurf=
   new G4OpticalSurface("photocath_opsurf",glisur,polished,
                        dielectric_metal);
G4double photocath_EFF[num]={1.,1.};
G4double photocath_REFL[num]={0.,0.};
G4MaterialPropertiesTable* photocath_mt = new G4MaterialPropertiesTable();
photocath_mt->AddProperty("EFFICIENCY",Ephoton,photocath_EFF,num);
photocath_mt->AddProperty("REFLECTIVITY",Ephoton,photocath_REFL,num);
photocath_opsurf->SetMaterialPropertiesTable(photocath_mt);
new G4LogicalSkinSurface("photocath_surf",photocath_log,photocath_opsurf);
```

La superficie de contacto del SiPM con la placa centelladora será la superficie sensible a los fotones, es decir el fotocátodo. Los fotoelectrones en el fotocátodo se generan de acuerdo con la eficiencia que este posea (QE por sus siglas en ingles de Quantum Efficiency). En la figura 7.17 se muestra las distribución del número de fotones opticos que impactan (hit) al fotocátodo del SiPM con cinco valores diferentes de QE, esto fue obtenido con un flujo de muones primarios de energía 1 GeV y recubrimiento de metal dieléctrico pulido.

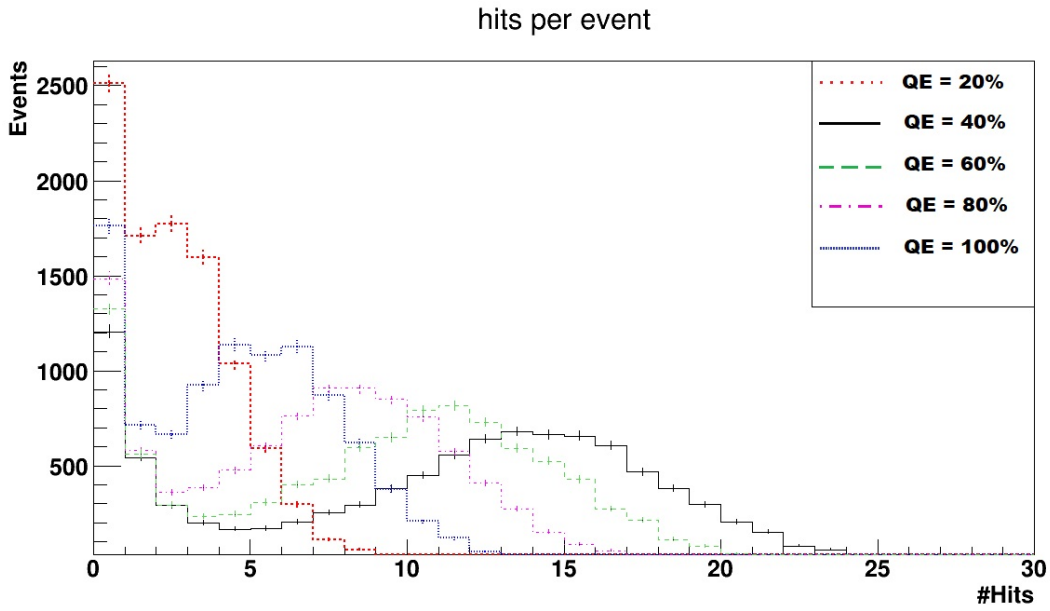

Figura 7.17: Distribución del número de fotones ópticos que impactan en el fotocátodo del SiPM para $QE = 20, 40, 60, 80, 100\,\%$

De la figura 7.17 se puede ver que si se establece una eficiencia del fotocátodo en el valor de $QE = 20\,\%$, se puede estudiar el proceso utilizando un enfoque de acertar o fallar. Por cada fotón óptico llegando al focátodo se genera un número aleatorio r y se compara con el $QE = 20\,\%$). Si $r \leq QE$ el fotón óptico puede producir un fotoelectrón en el tiempo t_i^{cath} (el tiempo de llegada del fotón al fotocátodo), de lo contrario no se produce ningún fotoelectrón. Alternativamente, el espectro QE mostrado en la anterior figura se puede usar para generar directamente los fotoelectrones en el fotocátodo.

Las trayectorias de las partículas generadas en la simulación pueden crear impactos en la placa de centelleo y el fotocátodo del SiPM que dentro de la aplicación creada en GEANT4 están definidos en las clases $ScintillatorHit$ y $PMTHit$, que a su vez son administradas por las clases de detector sensible (SD) $ScintillatorSD$ y $PMTSD$. Dentro de la simulación se crea un impacto de centelleo cuando la energía depositada por la trayectoria de una partícula no es nula. En este caso, se almacenan la energía depositada, la posición del vértice,

el tiempo, la identificación de la trayectoria y el tipo de partícula, junto con la información de identificación del centellador. Ahora bien, para los fotones ópticos que llegan al fotocátodo, se crea un impacto en el SiPM, entonces se almacena el número de fotones para cada proceso de generación (centelleo, Cherenkov, emisión WLS, etc. Ver figura 7.18), junto con la posición del fotón, el tiempo de llegada, la energía depositada en el SiPM (ver figura 7.19) y la identificación en el SiPM.

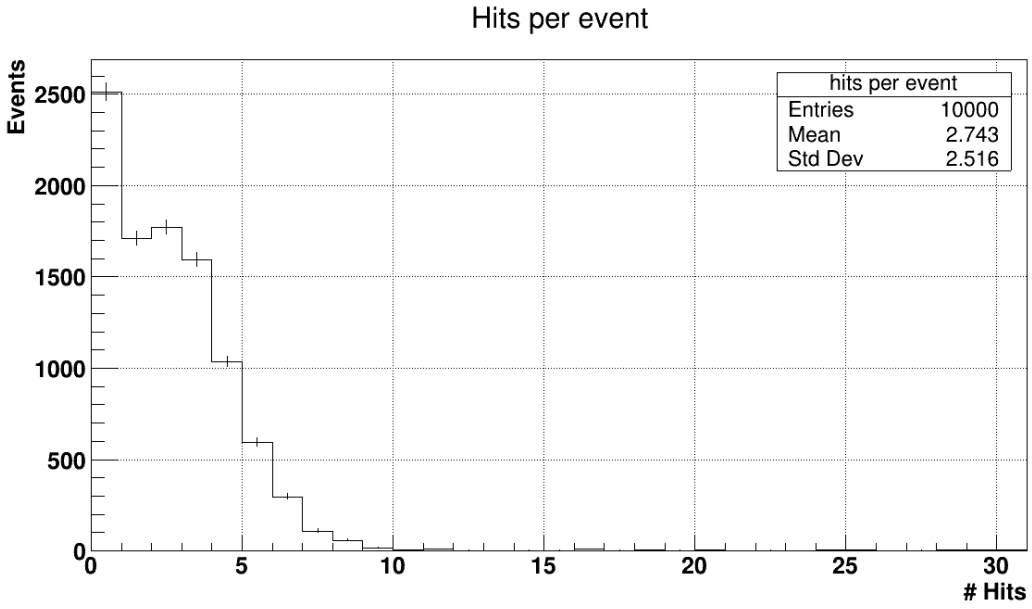

Figura 7.18: Número de impactos por evento generados por los fotones ópticos arrivando al fotocátodo del SiPM con $QE = 20\,\%$

7.4. Ejecución del programa

Para compilar la aplicación que contiene la simulación completa así como todos los resultados y gráficas correspondientes, GEANT4 necesita una función principal y mediante la clase $LXeRunAction.cc$ se inicializa el controlador $LXeRunManager.cc$ que es la encargada de cargar todas las clases ingresadas anteriormente y así poder usar $PhysicsList$ que se encarga de todos los procesos físicos e interacciones de la simulación. Para lanzarla se usa una

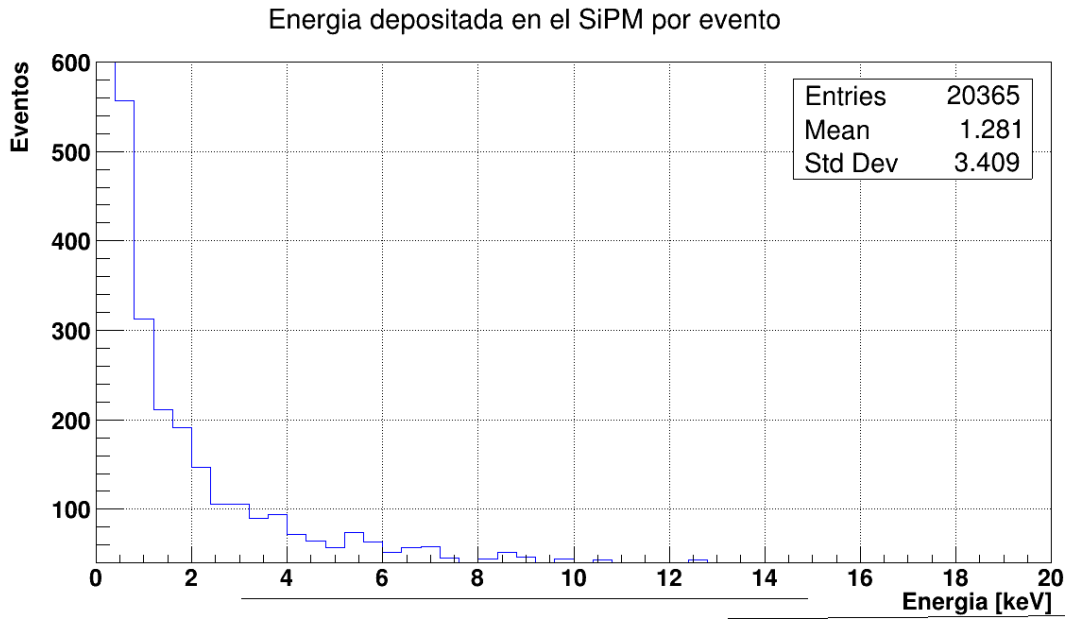

Figura 7.19: Energía depositada en el SiPM por los fotones ópticos que alcanzan a impactar en el fotocátodo.

terminal de linux, se ingresa al directorio $build$ en donde están todos las clases compiladas y se escribe: $\$./LXe\ cerenkov.mac$, esto lanza la simulación en modo ”batch”, ó también se puede escribir: $\$./LXe$, lo cuál lanza la simulación en modo ”interactivo”, es decir con visualización gráfica.

7.5. ROOT

Una vez termine de ejecutar la simulación en GEANT4[3] podemos disponer de la opción de presentar toda información recolectada en formato ROOT[35], además de una visualización para hacer los respectivos análisis. Como sabemos la clase que nos permite almacenar la información es $G4AnalisisManager$. En este caso únicamente se trabajó con la información obtenida por el archivo ROOT lo cuál permite ahorrar apreciablemente poder de computo, ya que en caso de hacer uso de la opción de visualización, es necesaria una alta capacidad de procesamiento de gráficos enfocada para las simulaciones de un gran número de partículas

generadas. En las figuras 7.3 y 7.4 se mostró el volumen físico para poder apreciar el entorno del mundo donde se va a trabajar, esto nos sirve para poder tener una representación geométrica de nuestro detector en uso.

Capítulo 8

Conclusiones y Discusión

Se ha estudiado cómo el volumen plástico centellador con un recubrimiento reflector y un fotodetector de metal dieléctrico pulido afecta las capacidades de un experimento basado en el uso de centelladores de plástico en el contexto de un experimento para física de partículas. El trabajo se ha centrado en un enfoque más general utilizando la herramienta de simulación GEANT4 altamente probada que permite considerar fácilmente un conjunto completo de propiedades y geometrías de materiales. Realizamos la simulación en GEANT4 de la respuesta de una geometría de material centellador plástico, teniendo en cuenta las propiedades ópticas de los medios del detector, así como todos los procesos físicos relevantes, y los fotones emitidos que se rastrearon explícitamente a lo largo de la geometría. La respuesta del SiPM se modeló para proporcionar observables directamente relacionados con los futuros datos de un prototipo de detector.

Se realizó una simulación detallada con el fin de considerar una elección en cuanto a diseño de estos detectores para su uso en el campo de la tomografía con muones. El uso de un material centellador permite emplear fotodetectores de Silicio (SiPM) para la colección de fotones y poder analizar la física óptica del detector con ayuda del software ROOT, de donde se obtuvo los siguientes resultados: En primer lugar se estudio el uso de un recubrimiento óptico para el detector, probando así el aumento en el número de reflexiones de los fotones emitidos, la alta ganancia de emisión de fotones ópticos dentro del material centellador y por ende el aumento en la eficiencia de los fotones colectados por el SiPM. También se analizó la fracción de fotones emitidos por el material centellador, que corresponde a cerca del $90\,\%$ que es absorbida en los limites del detector (como también por el SiPM) mientras que el resto se convierte en radiación de Cherenkov que termina atravesando, y por ende, escapando del detector. Finalmente, la respuesta que obtenemos del SiPM hace parte de la emisión de foto-

nes en el centellador que es absorbida en sus limites, la cuál depende de la eficiencia cuántica propia del SiPM, de donde tomando una eficiencia realista ($QE = 20\,\%$) se logra capturar una fracción de fotones que puede ser convertida posteriormente en señal eléctrónica.

Con referencia a los aspectos a discutir, en primera instancia se tiene el alcance de este trabajo realizado que hasta ahora ha ayudado en dos aspectos. En primer lugar, implementar y probar todas las herramientas de programación necesarias que nos proporciona GEANT4 para realizar simulaciones cuantitativas para la evaluación, análisis del transporte y captación de fotones ópticos en un volumen de material centellador; y en segundo lugar, cuantificar la importancia de varios aspectos, como las propiedades de reflexión del material de recubrimiento, la física óptica del material centellador y la eficiencia en la colección de fotones en el SiPM, todo esto ayudará para la selección del tipo y forma de un detector para aplicar tomografía con muones atmosféricos. Dependiendo del tamaño real que se elija para el detector, este puede requerir el uso de materiales de centelleo más gruesos o SiPM más eficientes para maximizar la cantidad de fotones transportados y detectados. Queda por fuera del objetivo de este trabajo, convertir la energía depositada por los fotones ópticos en el SiPM a señal electrónica, la implementación de esto último permitirá en un futuro poder comparar los resultados obtenidos con la simulación directamente con los datos generados por un detector de centelleo en el laboratorio.

Capítulo 9

Apéndice A

9.1. Código del programa

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

Headers (Carpeta include)

LXeActionInicialization.hh

```
#ifndef LXeActionInitialization_h
#define LXeActionInitialization_h 1

#include "G4VUserActionInitialization.hh"

class LXeDetectorConstruction;

class LXeActionInitialization : public G4VUserActionInitialization
{
 public:
  LXeActionInitialization(const LXeDetectorConstruction* det);
  ~LXeActionInitialization();

  void BuildForMaster() const override;
  void Build() const override;

 private:
  const LXeDetectorConstruction* fDetector;
};

#endif
```

LXeDetectorConstruction.hh

```
#ifndef LXeDetectorConstruction_h
#define LXeDetectorConstruction_h 1

#include "LXeDetectorMessenger.hh"

#include "G4Cache.hh"
#include "G4VUserDetectorConstruction.hh"

class LXeMainVolume;
class LXePMTSD;
class LXeScintSD;

class G4Box;
class G4Element;
class G4LogicalVolume;
class G4Material;
class G4MaterialPropertiesTable;
class G4Sphere;
class G4Tubs;
class G4VPhysicalVolume;

class LXeDetectorConstruction : public G4VUserDetectorConstruction
{
 public:
  LXeDetectorConstruction();
  ~LXeDetectorConstruction();

  G4VPhysicalVolume* Construct() override;
  void ConstructSDandField() override;

  // Functions to modify the geometry
  void SetDimensions(G4ThreeVector);
  void SetHousingThickness(G4double);
  void SetNX(G4int);
  void SetNY(G4int);
  void SetNZ(G4int);
  void SetPMTRadius(G4double);
  void SetDefaults();
  void SetSaveThreshold(G4int);
```

```
  // Get values
  G4int GetNX() const { return fNx; };
  G4int GetNY() const { return fNy; };
  G4int GetNZ() const { return fNz; };
  G4int GetSaveThreshold() const { return fSaveThreshold; };
  G4double GetScintX() const { return fScint_x; }
  G4double GetScintY() const { return fScint_y; }
  G4double GetScintZ() const { return fScint_z; }
  G4double GetHousingThickness() const { return fD_mtl; }
  G4double GetPMTRadius() const { return fOuterRadius_pmt; }
  G4double GetSlabZ() const { return fSlab_z; }

  void SetSphereOn(G4bool);
  static G4bool GetSphereOn() { return fSphereOn; }

  void SetHousingReflectivity(G4double);
  G4double GetHousingReflectivity() const { return fRefl; }

  void SetWLSSlabOn(G4bool b);
  G4bool GetWLSSlabOn() const { return fWLSslab; }

  void SetMainVolumeOn(G4bool b);
  G4bool GetMainVolumeOn() const { return fMainVolumeOn; }

  void SetNFibers(G4int n);
  G4int GetNFibers() const { return fNfibers; }

  void SetMainScintYield(G4double);
  void SetWLSScintYield(G4double);

private:
  void DefineMaterials();

  LXeDetectorMessenger* fDetectorMessenger;

  G4Box* fExperimentalHall_box;
  G4LogicalVolume* fExperimentalHall_log;
  G4VPhysicalVolume* fExperimentalHall_phys;
```

```
  // Materials & Elements
  G4Material* fLXe;
  G4Material* fAl;
  G4Element* fN;
  G4Element* fO;
  G4Material* fAir;
  G4Material* fVacuum;
  G4Element* fC;
  G4Element* fH;
  G4Material* fGlass;
  G4Material* fPstyrene;
  G4Material* fPMMA;
  G4Material* fPethylene1;
  G4Material* fPethylene2;

  // Geometry
  G4double fScint_x;
  G4double fScint_y;
  G4double fScint_z;
  G4double fD_mtl;
  G4int fNx;
  G4int fNy;
  G4int fNz;
  G4int fSaveThreshold;
  G4double fOuterRadius_pmt;
  G4int fNfibers;
  static G4bool fSphereOn;
  G4double fRefl;
  G4bool fWLSslab;
  G4bool fMainVolumeOn;
  G4double fSlab_z;

  LXeMainVolume* fMainVolume;

  G4MaterialPropertiesTable* fLXe_mt;
  G4MaterialPropertiesTable* fMPTPStyrene;

  // Sensitive Detectors
  G4Cache<LXeScintSD*> fScint_SD;
  G4Cache<LXePMTSD*> fPmt_SD;
};

#endif
```

LXeDetectorMessenger.hh

```
#ifndef LXeDetectorMessenger_h
#define LXeDetectorMessenger_h 1

#include "globals.hh"
#include "G4UImessenger.hh"

class LXeDetectorConstruction;

class G4UIcmdWithABool;
class G4UIcmdWithADouble;
class G4UIcmdWithADoubleAndUnit;
class G4UIcmdWithAnInteger;
class G4UIcmdWith3VectorAndUnit;
class G4UIcommand;
class G4UIdirectory;
class LXeDetectorMessenger : public G4UImessenger
{
 public:
  LXeDetectorMessenger(LXeDetectorConstruction*);
  ~LXeDetectorMessenger();

  void SetNewValue(G4UIcommand*, G4String) override;

 private:
  LXeDetectorConstruction* fLXeDetector;
  G4UIdirectory* fDetectorDir;
  G4UIdirectory* fVolumesDir;
  G4UIcmdWith3VectorAndUnit* fDimensionsCmd;
  G4UIcmdWithADoubleAndUnit* fHousingThicknessCmd;
  G4UIcmdWithADoubleAndUnit* fPmtRadiusCmd;
  G4UIcmdWithAnInteger* fNxCmd;
  G4UIcmdWithAnInteger* fNyCmd;
  G4UIcmdWithAnInteger* fNzCmd;
  G4UIcmdWithABool* fSphereCmd;
  G4UIcmdWithADouble* fReflectivityCmd;
  G4UIcmdWithABool* fWlsCmd;
  G4UIcmdWithABool* fLxeCmd;
  G4UIcmdWithAnInteger* fNFibersCmd;
  G4UIcommand* fDefaultsCmd;
  G4UIcmdWithADouble* fMainScintYield;
  G4UIcmdWithADouble* fWLSScintYield;
  G4UIcmdWithAnInteger* fSaveThresholdCmd;
};
#endif
```

LXeEventAction.hh

```
#ifndef LXeEventAction_h
#define LXeEventAction_h 1

#include "LXeEventMessenger.hh"

#include "globals.hh"
#include "G4ThreeVector.hh"
#include "G4UserEventAction.hh"

class G4Event;
class LXeDetectorConstruction;

class LXeEventAction : public G4UserEventAction
{
 public:
  LXeEventAction(const LXeDetectorConstruction*);
  ~LXeEventAction();

 public:
  void BeginOfEventAction(const G4Event*) override;
  void EndOfEventAction(const G4Event*) override;

  void SetEventVerbose(G4int v) { fVerbose = v; }

  void SetPMTThreshold(G4int t) { fPMTThreshold = t; }

  void SetForceDrawPhotons(G4bool b) { fForcedrawphotons = b; }
  void SetForceDrawNoPhotons(G4bool b) { fForcenophotons = b; }

  void IncPhotonCount_Scint() { ++fPhotonCount_Scint; }
  void IncPhotonCount_Ceren() { ++fPhotonCount_Ceren; }
  void IncEDep(G4double dep) { fTotE += dep; }
    void IncEDep2(G4double dep2) { fTotE3 += dep2; }
  void IncAbsorption() { ++fAbsorptionCount; }
  void IncBoundaryAbsorption() { ++fBoundaryAbsorptionCount; }
  void IncHitCount(G4int i = 1) { fHitCount += i; }

  void SetEWeightPos(const G4ThreeVector& p) { fEWeightPos = p; }
    void SetEWeightPos2(const G4ThreeVector& p) { fEWeightPos2 = p; }
  void SetReconPos(const G4ThreeVector& p) { fReconPos = p; }
    void SetReconPos2(const G4ThreeVector& p) { fReconPos2 = p; }
  void SetConvPos(const G4ThreeVector& p)
    //void SetConvPos2(const G4ThreeVector& p)
```

```
  {
    fConvPos    = p;
    fConvPosSet = true;
        fConvPos2    = p;
        fConvPosSet2 = true;
  }
  void SetPosMax(const G4ThreeVector& p, G4double edep, G4double edep2)
  {
    fPosMax  = p;
    fEdepMax = edep;
        fPosMax2  = p;
        fEdepMax2 = edep2;
  }

  G4int GetPhotonCount_Scint() const { return fPhotonCount_Scint; }
  G4int GetPhotonCount_Ceren() const { return fPhotonCount_Ceren; }
  G4int GetHitCount() const { return fHitCount; }
  G4double GetEDep() const { return fTotE; }
    G4double GetEDep2() const { return fTotE3; }
  G4int GetAbsorptionCount() const { return fAbsorptionCount; }
  G4int GetBoundaryAbsorptionCount() const { return fBoundaryAbsorptionCount; }

  G4ThreeVector GetEWeightPos() { return fEWeightPos; }
    G4ThreeVector GetEWeightPos2() { return fEWeightPos2; }
  G4ThreeVector GetReconPos() { return fReconPos; }
  G4ThreeVector GetConvPos() { return fConvPos; }
  G4ThreeVector GetPosMax() { return fPosMax; }
   G4ThreeVector GetPosMax2() { return fPosMax2; }
  G4double GetEDepMax() { return fEdepMax; }
    G4double GetEDepMax2() { return fEdepMax2; }
  G4double IsConvPosSet() { return fConvPosSet; }
    G4double IsConvPosSet2() { return fConvPosSet; }

  // Gets the total photon count produced
  G4int GetPhotonCount() { return fPhotonCount_Scint + fPhotonCount_Ceren; }

  void IncPMTSAboveThreshold() { ++fPMTsAboveThreshold; }
  G4int GetPMTSAboveThreshold() { return fPMTsAboveThreshold; }

private:
  LXeEventMessenger* fEventMessenger;
  const LXeDetectorConstruction* fDetector;
```

```
  G4int fScintCollID;
  G4int fPMTCollID;

  G4int fVerbose;

  G4int fPMTThreshold;

  G4bool fForcedrawphotons;
  G4bool fForcenophotons;

  G4int fHitCount;
  G4int fPhotonCount_Scint;
  G4int fPhotonCount_Ceren;
  G4int fAbsorptionCount;
  G4int fBoundaryAbsorptionCount;

  G4double fTotE;
    G4double fTotE3;

  // These only have meaning if totE > 0
  // If totE = 0 then these won't be set by EndOfEventAction
  G4ThreeVector fEWeightPos;
    G4ThreeVector fEWeightPos2;
  G4ThreeVector fReconPos;
    G4ThreeVector fReconPos2;
  G4ThreeVector fConvPos;
    G4ThreeVector fConvPos2;
  G4bool fConvPosSet;
    G4bool fConvPosSet2;
  G4ThreeVector fPosMax;
    G4ThreeVector fPosMax2;
  G4double fEdepMax;
    G4double fEdepMax2;
  G4int fPMTsAboveThreshold;
};

#endif
```

LXeEventMessenger.hh

```
#ifndef LXeEventMessenger_h
#define LXeEventMessenger_h 1

#include "globals.hh"
#include "G4UImessenger.hh"

class LXeEventAction;
class G4UIcmdWithABool;
class G4UIcmdWithAnInteger;

class LXeEventMessenger : public G4UImessenger
{
 public:
  LXeEventMessenger(LXeEventAction*);
  ~LXeEventMessenger();

  void SetNewValue(G4UIcommand*, G4String) override;

 private:
  LXeEventAction* fLXeEvent;
  G4UIcmdWithAnInteger* fVerboseCmd;
  G4UIcmdWithAnInteger* fPmtThresholdCmd;
  G4UIcmdWithABool* fForceDrawPhotonsCmd;
  G4UIcmdWithABool* fForceDrawNoPhotonsCmd;
};

#endif
```

LXeHistoManager.hh

```
#ifndef LXeHistoManager_h
#define LXeHistoManager_h 1

#include "globals.hh"

#include "g4root.hh"
//#include "g4xml.hh"
//#include "g4csv.hh"

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

class LXeHistoManager
{
 public:
  LXeHistoManager();
  ~LXeHistoManager();

 private:
  void Book();
  G4String fFileName;
};

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

#endif
```

LXeMainVolume.hh

```
#ifndef LXeMainVolume_h
#define LXeMainVolume_h 1

#include "LXeDetectorConstruction.hh"

#include "G4PVPlacement.hh"

class G4Box;
class G4LogicalVolume;
class G4Sphere;
class G4Tubs;

class LXeMainVolume : public G4PVPlacement
{
 public:
  LXeMainVolume(G4RotationMatrix* pRot, const G4ThreeVector& tlate,
                G4LogicalVolume* pMotherLogical, G4bool pMany, G4int pCopyNo,
                LXeDetectorConstruction* c);

  G4LogicalVolume* GetLogPhotoCath() { return fPhotocath_log; }
  G4LogicalVolume* GetLogScint() { return fScint_log; }

  std::vector<G4ThreeVector> GetPmtPositions() { return fPmtPositions; }

 private:
  void VisAttributes();
  void SurfaceProperties();

  void PlacePMTs(G4LogicalVolume* pmt_Log, G4RotationMatrix* rot, G4double& a,
                 G4double& b, G4double da, G4double db, G4double amin,
                 G4double bmin, G4int na, G4int nb, G4double& x, G4double& y,
                 G4double& z, G4int& k);

  void CopyValues();

  LXeDetectorConstruction* fConstructor;

  G4double fScint_x;
  G4double fScint_y;
  G4double fScint_z;
  G4double fD_mtl;
  G4int fNx;
  G4int fNy;
  G4int fNz;
  G4double fOuterRadius_pmt;
```

```
  G4bool fSphereOn;
  G4double fRefl;

  // Basic Volumes
  //
  G4Box* fScint_box;
  G4Box* fHousing_box;
  G4Box* fPmt;
  G4Box* fPhotocath;
  G4Sphere* fSphere;

  // Logical volumes
  //
  G4LogicalVolume* fScint_log;
  G4LogicalVolume* fHousing_log;
  G4LogicalVolume* fPmt_log;
  G4LogicalVolume* fPhotocath_log;
  G4LogicalVolume* fSphere_log;

  // Sensitive Detectors positions
  std::vector<G4ThreeVector> fPmtPositions;
};

#endif
```

LXePMTHit.hh

```
#ifndef LXePMTHit_h
#define LXePMTHit_h 1

#include "G4Allocator.hh"
#include "G4LogicalVolume.hh"
#include "G4THitsCollection.hh"
#include "G4VHit.hh"
#include "G4VPhysicalVolume.hh"

class LXePMTHit : public G4VHit
{
 public:
  LXePMTHit();
  LXePMTHit(const LXePMTHit& right);
    LXePMTHit(G4VPhysicalVolume* pVol);
  ~LXePMTHit();

  const LXePMTHit& operator=(const LXePMTHit& right);
  G4bool operator==(const LXePMTHit& right) const;

  inline void* operator new(size_t);
  inline void operator delete(void* aHit);

    inline void SetEdep2(G4double de) { fEdep2 = de; }
    inline void AddEdep2(G4double de) { fEdep2 += de; }
    inline G4double GetEdep2() { return fEdep2; }


  virtual void Draw();
  virtual void Print();

  inline void SetDrawit(G4bool b) { fDrawit = b; }
  inline G4bool GetDrawit() { return fDrawit; }

  inline void IncPhotonCount() { ++fPhotons; }
  inline G4int GetPhotonCount() { return fPhotons; }

  inline void SetPMTNumber(G4int n) { fPmtNumber = n; }
  inline G4int GetPMTNumber() { return fPmtNumber; }

  inline void SetPMTPhysVol(G4VPhysicalVolume* physVol)
  {
    this->fPhysVol = physVol;
  }
```

```
    //inline const G4VPhysicalVolume* GetPMTPhysVol2() { return fPhysVol; }

  inline G4VPhysicalVolume* GetPMTPhysVol() { return fPhysVol; }

  inline void SetPMTPos(G4double x, G4double y, G4double z)
  {
    fPos = G4ThreeVector(x, y, z);
  }

     inline void SetPMTPos2(G4ThreeVector xyz) { fPos = xyz; }

  inline G4ThreeVector GetPMTPos() { return fPos; }



 private:

  G4int fPmtNumber;
  G4int fPhotons;
      G4double fEdep2;
  G4ThreeVector fPos;
  G4VPhysicalVolume* fPhysVol;
  G4bool fDrawit;

};

typedef G4THitsCollection<LXePMTHit> LXePMTHitsCollection;

extern G4ThreadLocal G4Allocator<LXePMTHit>* LXePMTHitAllocator;

inline void* LXePMTHit::operator new(size_t)
{
  if(!LXePMTHitAllocator)
    LXePMTHitAllocator = new G4Allocator<LXePMTHit>;
  return (void*) LXePMTHitAllocator->MallocSingle();
}

inline void LXePMTHit::operator delete(void* aHit)
{
  LXePMTHitAllocator->FreeSingle((LXePMTHit*) aHit);
}

#endif
```

LXePMTSD.hh

```
#ifndef LXePMTSD_h
#define LXePMTSD_h 1

#include "LXePMTHit.hh"

#include "G4VSensitiveDetector.hh"

#include <vector>

class G4DataVector;
class G4HCofThisEvent;
class G4Step;

class LXePMTSD : public G4VSensitiveDetector
{
 public:
  LXePMTSD(G4String name);
  ~LXePMTSD();

  void Initialize(G4HCofThisEvent*) override;
  G4bool ProcessHits(G4Step* aStep, G4TouchableHistory*) override;

  // A version of processHits active on boundary
  G4bool ProcessHits_boundary(const G4Step*, G4TouchableHistory*);

  // Initialize the arrays to store pmt possitions
  inline void InitPMTs()
  {
    if(fPMTPositionsX)
      delete fPMTPositionsX;
    if(fPMTPositionsY)
      delete fPMTPositionsY;
    if(fPMTPositionsZ)
      delete fPMTPositionsZ;
    fPMTPositionsX = new G4DataVector();
    fPMTPositionsY = new G4DataVector();
    fPMTPositionsZ = new G4DataVector();
  }

  // Store a pmt position
  void SetPmtPositions(const std::vector<G4ThreeVector>& positions);

 private:
  LXePMTHitsCollection* fPMTHitCollection;
```

```
  G4DataVector* fPMTPositionsX;
  G4DataVector* fPMTPositionsY;
  G4DataVector* fPMTPositionsZ;

  G4int fHitCID;
};

#endif
```

LXePrimaryGeneratorAction.hh

```
#ifndef LXePrimaryGeneratorAction_h
#define LXePrimaryGeneratorAction_h 1

#include "G4VUserPrimaryGeneratorAction.hh"

class G4ParticleGun;
class G4Event;

class LXePrimaryGeneratorAction : public G4VUserPrimaryGeneratorAction
{
 public:
  LXePrimaryGeneratorAction();
  ~LXePrimaryGeneratorAction();

  void GeneratePrimaries(G4Event* anEvent) override;

 private:
  G4ParticleGun* fParticleGun;
};

#endif
```

LXeRun.hh

```
#ifndef LXeRun_h
#define LXeRun_h 1

#include "globals.hh"
#include "G4Run.hh"

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

class LXeRun : public G4Run
{
 public:
  LXeRun();
  ~LXeRun();

  void IncPhotonCount_Scint(G4int count)
  {
    fPhotonCount_Scint += count;
    fPhotonCount_Scint2 += count * count;
  }
  void IncPhotonCount_Ceren(G4int count)
  {
    fPhotonCount_Ceren += count;
    fPhotonCount_Ceren2 += count * count;
  }
  void IncEDep(G4double dep)
  {
    fTotE += dep;
    fTotE2 += dep * dep;
  }
  void IncEDep2(G4double dep2)   //nuevo
  {
    fTotE3 += dep2;
    fTotE4 += dep2 * dep2;
  }
  void IncAbsorption(G4int count)
  {
    fAbsorptionCount += count;
    fAbsorptionCount2 += count * count;
  }
  void IncBoundaryAbsorption(G4int count)
  {
    fBoundaryAbsorptionCount += count;
    fBoundaryAbsorptionCount2 += count * count;
  }
```

```
  void IncHitCount(G4int count)
  {
    fHitCount += count;
    fHitCount2 += count * count;
  }
  void IncHitsAboveThreshold(G4int count)
  {
    fPMTsAboveThreshold += count;
    fPMTsAboveThreshold2 += count * count;
  }

  void Merge(const G4Run* run) override;

  void EndOfRun();

 private:
  G4int fHitCount;
  G4int fHitCount2;
  G4int fPhotonCount_Scint;
  G4int fPhotonCount_Scint2;
  G4int fPhotonCount_Ceren;
  G4int fPhotonCount_Ceren2;
  G4int fAbsorptionCount;
  G4int fAbsorptionCount2;
  G4int fBoundaryAbsorptionCount;
  G4int fBoundaryAbsorptionCount2;
  G4int fPMTsAboveThreshold;
  G4int fPMTsAboveThreshold2;

  G4double fTotE;
  G4double fTotE2;
    G4double fTotE3;
    G4double fTotE4;
};

#endif  // LXeRun_h
```

LXeRunAction.hh

```
#include "G4UserRunAction.hh"

#ifndef LXeRunAction_h
#  define LXeRunAction_h 1

class LXeRun;
class LXeHistoManager;

class G4Run;

class LXeRunAction : public G4UserRunAction
{
 public:
  LXeRunAction();
  ~LXeRunAction();

  G4Run* GenerateRun() override;
  void BeginOfRunAction(const G4Run*) override;
  void EndOfRunAction(const G4Run*) override;

 private:
  LXeRun* fRun;
  LXeHistoManager* fHistoManager;
};

#endif
```

LXeScintHit.hh

```
#ifndef LXeScintHit_h
#define LXeScintHit_h 1

#include "G4VHit.hh"
#include "G4THitsCollection.hh"
#include "G4Allocator.hh"
#include "G4ThreeVector.hh"
#include "G4VPhysicalVolume.hh"

class LXeScintHit : public G4VHit
{
 public:
  LXeScintHit();
  LXeScintHit(G4VPhysicalVolume* pVol);
  ~LXeScintHit();

  LXeScintHit(const LXeScintHit& right);
  const LXeScintHit& operator=(const LXeScintHit& right);
  G4bool operator==(const LXeScintHit& right) const;

  inline void* operator new(size_t);
  inline void operator delete(void* aHit);

  inline void SetEdep(G4double de) { fEdep = de; }
  inline void AddEdep(G4double de) { fEdep += de; }
  inline G4double GetEdep() { return fEdep; }

  inline void SetPos(G4ThreeVector xyz) { fPos = xyz; }
  inline G4ThreeVector GetPos() { return fPos; }

  inline const G4VPhysicalVolume* GetPhysV() { return fPhysVol; }

 private:
  G4double fEdep;
  G4ThreeVector fPos;
  const G4VPhysicalVolume* fPhysVol;
};

typedef G4THitsCollection<LXeScintHit> LXeScintHitsCollection;

extern G4ThreadLocal G4Allocator<LXeScintHit>* LXeScintHitAllocator;
```

```
inline void* LXeScintHit::operator new(size_t)
{
  if(!LXeScintHitAllocator)
    LXeScintHitAllocator = new G4Allocator<LXeScintHit>;
  return (void*) LXeScintHitAllocator->MallocSingle();
}

inline void LXeScintHit::operator delete(void* aHit)
{
  LXeScintHitAllocator->FreeSingle((LXeScintHit*) aHit);
}

#endif
```

LXeScintSD.hh

```
#ifndef LXeScintSD_h
#define LXeScintSD_h 1

#include "LXeScintHit.hh"

#include "G4VSensitiveDetector.hh"

class G4Step;
class G4HCofThisEvent;

class LXeScintSD : public G4VSensitiveDetector
{
 public:
  LXeScintSD(G4String name);
  ~LXeScintSD();

  void Initialize(G4HCofThisEvent*) override;
  G4bool ProcessHits(G4Step* aStep, G4TouchableHistory*) override;

 private:
  LXeScintHitsCollection* fScintCollection;
  G4int fHitsCID;
};

#endif
```

LXeStackingAction.hh

```
#ifndef LXeStackingAction_h
#define LXeStackingAction_h 1

#include "G4UserStackingAction.hh"

class LXeEventAction;

class LXeStackingAction : public G4UserStackingAction
{
 public:
  LXeStackingAction(LXeEventAction*);
  ~LXeStackingAction();

  G4ClassificationOfNewTrack ClassifyNewTrack(const G4Track* aTrack) override;

 private:
  LXeEventAction* fEventAction;
};

#endif
```

LXeSteppingAction.hh

```
#ifndef LXeSteppingAction_h
#define LXeSteppingACtion_h 1

#include "globals.hh"
#include "G4OpBoundaryProcess.hh"
#include "G4UserSteppingAction.hh"

class LXeEventAction;
class LXeTrackingAction;
class LXeSteppingMessenger;

class LXeSteppingAction : public G4UserSteppingAction
{
 public:
  LXeSteppingAction(LXeEventAction*);
  ~LXeSteppingAction();

  void UserSteppingAction(const G4Step*) override;

  void SetOneStepPrimaries(G4bool b) { fOneStepPrimaries = b; }
  G4bool GetOneStepPrimaries() { return fOneStepPrimaries; }

 private:
  G4bool fOneStepPrimaries;
  LXeSteppingMessenger* fSteppingMessenger;
  LXeEventAction* fEventAction;

  G4OpBoundaryProcessStatus fExpectedNextStatus;
};

#endif
```

LXeSteppingMessenger.hh

```
#ifndef LXeSteppingMessenger_h
#define LXeSteppingMessenger_h 1

#include "G4UImessenger.hh"

class LXeSteppingAction;

class G4UIcmdWithABool;

class LXeSteppingMessenger : public G4UImessenger
{
 public:
  LXeSteppingMessenger(LXeSteppingAction*);
  ~LXeSteppingMessenger();

  void SetNewValue(G4UIcommand*, G4String) override;

 private:
  LXeSteppingAction* fStepping;
  G4UIcmdWithABool* fOneStepPrimariesCmd;
};

#endif
```

LXeTrackingAction.hh

```
#ifndef LXeTrackingAction_h
#define LXeTrackingAction_h 1

#include "globals.hh"
#include "G4UserTrackingAction.hh"

class LXeTrackingAction : public G4UserTrackingAction
{
 public:
  LXeTrackingAction();
  ~LXeTrackingAction(){};

  void PreUserTrackingAction(const G4Track*) override;
  void PostUserTrackingAction(const G4Track*) override;

 private:
};

#endif
```

LXeUserTrackInformation.hh

```
#include "G4VUserTrackInformation.hh"
#include "globals.hh"

#ifndef LXeUserTrackInformation_h
#  define LXeUserTrackInformation_h 1

enum LXeTrackStatus
{
  active           = 1,
  hitPMT           = 2,
  absorbed         = 4,
  boundaryAbsorbed = 8,
  inactive         = 14
};

class LXeUserTrackInformation : public G4VUserTrackInformation
{
 public:
  LXeUserTrackInformation();
  ~LXeUserTrackInformation();

  // Sets the track status to s (does not check validity of flags)
  void SetTrackStatusFlags(int s) { fStatus = s; }
  // Does a smart add of track status flags (disabling old flags that conflict)
  // If s conflicts with itself it will not be detected
  void AddTrackStatusFlag(int s);

  int GetTrackStatus() const { return fStatus; }

  void IncReflections() { ++fReflections; }
  G4int GetReflectionCount() const { return fReflections; }

  void SetForceDrawTrajectory(G4bool b) { fForcedraw = b; }
  G4bool GetForceDrawTrajectory() { return fForcedraw; }

  inline virtual void Print() const {};

 private:
  int fStatus;
  G4int fReflections;
  G4bool fForcedraw;
};

#endif
```

Clases usuario (Carpeta src)

LXeActionInitialization.cc

```
#include "LXeActionInitialization.hh"

#include "LXeDetectorConstruction.hh"
#include "LXeEventAction.hh"
#include "LXePrimaryGeneratorAction.hh"
#include "LXeRunAction.hh"
#include "LXeStackingAction.hh"
#include "LXeSteppingAction.hh"
#include "LXeTrackingAction.hh"

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

LXeActionInitialization::LXeActionInitialization(
  const LXeDetectorConstruction* det)
  : G4VUserActionInitialization()
  , fDetector(det)
{}

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

LXeActionInitialization::~LXeActionInitialization() {}

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

void LXeActionInitialization::BuildForMaster() const
{
  SetUserAction(new LXeRunAction());
}

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

void LXeActionInitialization::Build() const
{
  SetUserAction(new LXePrimaryGeneratorAction());

  LXeEventAction* eventAction = new LXeEventAction(fDetector);
  SetUserAction(eventAction);
  SetUserAction(new LXeStackingAction(eventAction));

  SetUserAction(new LXeRunAction());
  SetUserAction(new LXeTrackingAction());
  SetUserAction(new LXeSteppingAction(eventAction));
}
```

LXeDetectorConstruction.cc

```
#include "LXeDetectorConstruction.hh"

#include "LXeDetectorMessenger.hh"
#include "LXeMainVolume.hh"
#include "LXePMTSD.hh"
#include "LXeScintSD.hh"
#include "LXeWLSSlab.hh"

#include "globals.hh"
#include "G4Box.hh"
#include "G4GeometryManager.hh"
#include "G4LogicalBorderSurface.hh"
#include "G4LogicalSkinSurface.hh"
#include "G4LogicalVolume.hh"
#include "G4LogicalVolumeStore.hh"
#include "G4Material.hh"
#include "G4MaterialTable.hh"
#include "G4OpticalSurface.hh"
#include "G4PhysicalConstants.hh"
#include "G4PhysicalVolumeStore.hh"
#include "G4PVPlacement.hh"
#include "G4RunManager.hh"
#include "G4SDManager.hh"
#include "G4SolidStore.hh"
#include "G4Sphere.hh"
#include "G4SystemOfUnits.hh"
#include "G4ThreeVector.hh"
#include "G4Tubs.hh"
#include "G4UImanager.hh"
#include "G4VisAttributes.hh"

G4bool LXeDetectorConstruction::fSphereOn = true;

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

LXeDetectorConstruction::LXeDetectorConstruction()
  : fLXe_mt(nullptr)
  , fMPTPStyrene(nullptr)
{
  fExperimentalHall_box  = nullptr;
  fExperimentalHall_log  = nullptr;
  fExperimentalHall_phys = nullptr;
```

```
  fLXe = fAl = fAir = fVacuum = fGlass = nullptr;
  fPstyrene = fPMMA = fPethylene1 = fPethylene2 = nullptr;

  fN = fO = fC = fH = nullptr;

  fSaveThreshold = 0;
  SetDefaults();

  DefineMaterials();
  fDetectorMessenger = new LXeDetectorMessenger(this);
}

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

LXeDetectorConstruction::~LXeDetectorConstruction()
{
  if(fMainVolume)
  {
    delete fMainVolume;
  }
  delete fLXe_mt;
  delete fDetectorMessenger;
  delete fMPTPStyrene;
}

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

void LXeDetectorConstruction::DefineMaterials()
{
  G4double a;  // atomic mass
  G4double z;  // atomic number
  G4double density;

  G4int polyPMMA = 1;
  G4int nC_PMMA  = 3 + 2 * polyPMMA;
  G4int nH_PMMA  = 6 + 2 * polyPMMA;

  G4int polyeth = 1;
  G4int nC_eth  = 2 * polyeth;
  G4int nH_eth  = 4 * polyeth;

  //***Elements
  fH = new G4Element("H", "H", z = 1., a = 1.01 * g / mole);
  fC = new G4Element("C", "C", z = 6., a = 12.01 * g / mole);
```

```
fN = new G4Element("N", "N", z = 7., a = 14.01 * g / mole);
fO = new G4Element("O", "O", z = 8., a = 16.00 * g / mole);

//***Materials
// Cintillation
fLXe = G4NistManager::Instance()->FindOrBuildMaterial("G4_PLASTIC_SC_VINYLTOLUENE");

//Silicio
fAl = G4NistManager::Instance()->FindOrBuildMaterial("G4_Si");

// Vacuum
fVacuum = new G4Material("Vacuum", z = 1., a = 1.01 * g / mole,
                         density = universe_mean_density, kStateGas,
                         0.1 * kelvin, 1.e-19 * pascal);
// Air
fAir = new G4Material("Air", density = 1.29 * mg / cm3, 2);
fAir->AddElement(fN, 70 * perCent);
fAir->AddElement(fO, 30 * perCent);
// Glass
fGlass = new G4Material("Glass", density = 1.032 * g / cm3, 2);
fGlass->AddElement(fC, 91.533 * perCent);
fGlass->AddElement(fH, 8.467 * perCent);
// Polystyrene
fPstyrene = new G4Material("Polystyrene", density = 1.03 * g / cm3, 2);
fPstyrene->AddElement(fC, 8);
fPstyrene->AddElement(fH, 8);
// Fiber(PMMA)
fPMMA = new G4Material("PMMA", density = 1190. * kg / m3, 3);
fPMMA->AddElement(fH, nH_PMMA);
fPMMA->AddElement(fC, nC_PMMA);
fPMMA->AddElement(fO, 2);
// Cladding(polyethylene)
fPethylene1 = new G4Material("Pethylene1", density = 1200. * kg / m3, 2);
fPethylene1->AddElement(fH, nH_eth);
fPethylene1->AddElement(fC, nC_eth);
// Double cladding(flourinated polyethylene)
fPethylene2 = new G4Material("Pethylene2", density = 1400. * kg / m3, 2);
fPethylene2->AddElement(fH, nH_eth);
fPethylene2->AddElement(fC, nC_eth);

//***Material properties tables

std::vector<G4double> lxe_Energy = { 7.0 * eV, 7.07 * eV, 7.14 * eV };
```

```
std::vector<G4double> lxe_SCINT = { 0.1, 1.0, 0.1 };
std::vector<G4double> lxe_RIND  = { 1.59, 1.57, 1.54 };
std::vector<G4double> lxe_ABSL  = { 35. * cm, 35. * cm, 35. * cm };
fLXe_mt = new G4MaterialPropertiesTable();
fLXe_mt->AddProperty("SCINTILLATIONCOMPONENT1", lxe_Energy, lxe_SCINT);
fLXe_mt->AddProperty("SCINTILLATIONCOMPONENT2", lxe_Energy, lxe_SCINT);
fLXe_mt->AddProperty("RINDEX", lxe_Energy, lxe_RIND);
fLXe_mt->AddProperty("ABSLENGTH", lxe_Energy, lxe_ABSL);
fLXe_mt->AddConstProperty("SCINTILLATIONYIELD", 12000. / MeV);
fLXe_mt->AddConstProperty("RESOLUTIONSCALE", 1.0);
fLXe_mt->AddConstProperty("SCINTILLATIONTIMECONSTANT1", 20. * ns);
fLXe_mt->AddConstProperty("SCINTILLATIONTIMECONSTANT2", 45. * ns);
fLXe_mt->AddConstProperty("SCINTILLATIONYIELD1", 1.0);
fLXe_mt->AddConstProperty("SCINTILLATIONYIELD2", 0.0);
fLXe->SetMaterialPropertiesTable(fLXe_mt);

// Set the Birks Constant for the LXe scintillator
fLXe->GetIonisation()->SetBirksConstant(0.126 * mm / MeV);

std::vector<G4double> glass_RIND      = { 1.49, 1.49, 1.49 };
std::vector<G4double> glass_AbsLength = { 420. * cm, 420. * cm, 420. * cm };
G4MaterialPropertiesTable* glass_mt   = new G4MaterialPropertiesTable();
glass_mt->AddProperty("ABSLENGTH", lxe_Energy, glass_AbsLength);
glass_mt->AddProperty("RINDEX", lxe_Energy, glass_RIND);
fGlass->SetMaterialPropertiesTable(glass_mt);

std::vector<G4double> vacuum_Energy  = { 2.0 * eV, 7.0 * eV, 7.14 * eV };
std::vector<G4double> vacuum_RIND    = { 1., 1., 1. };
G4MaterialPropertiesTable* vacuum_mt = new G4MaterialPropertiesTable();
vacuum_mt->AddProperty("RINDEX", vacuum_Energy, vacuum_RIND);
fVacuum->SetMaterialPropertiesTable(vacuum_mt);
fAir->SetMaterialPropertiesTable(vacuum_mt);  // Give air the same rindex

std::vector<G4double> wls_Energy = { 2.00 * eV, 2.87 * eV, 2.90 * eV,
                                     3.47 * eV };

std::vector<G4double> rIndexPstyrene = { 1.5, 1.5, 1.5, 1.5 };
std::vector<G4double> absorption1    = { 2. * cm, 2. * cm, 2. * cm, 2. * cm };
std::vector<G4double> scintilFast    = { 0.0, 0.0, 1.0, 1.0 };
fMPTPStyrene = new G4MaterialPropertiesTable();
fMPTPStyrene->AddProperty("RINDEX", wls_Energy, rIndexPstyrene);
fMPTPStyrene->AddProperty("ABSLENGTH", wls_Energy, absorption1);
fMPTPStyrene->AddProperty("SCINTILLATIONCOMPONENT1", wls_Energy, scintilFast);
fMPTPStyrene->AddConstProperty("SCINTILLATIONYIELD", 10. / keV);
```

```
  fMPTPStyrene->AddConstProperty("RESOLUTIONSCALE", 1.0);
  fMPTPStyrene->AddConstProperty("SCINTILLATIONTIMECONSTANT1", 10. * ns);
  fPstyrene->SetMaterialPropertiesTable(fMPTPStyrene);

  // Set the Birks Constant for the Polystyrene scintillator
  fPstyrene->GetIonisation()->SetBirksConstant(0.126 * mm / MeV);

  std::vector<G4double> RefractiveIndexFiber = { 1.6, 1.6, 1.6, 1.6 };
  std::vector<G4double> AbsFiber    = { 9.0 * m, 9.0 * m, 0.1 * mm, 0.1 * mm };
  std::vector<G4double> EmissionFib = { 1.0, 1.0, 0.0, 0.0 };
  G4MaterialPropertiesTable* fiberProperty = new G4MaterialPropertiesTable();
  fiberProperty->AddProperty("RINDEX", wls_Energy, RefractiveIndexFiber);
  fiberProperty->AddProperty("WLSABSLENGTH", wls_Energy, AbsFiber);
  fiberProperty->AddProperty("WLSCOMPONENT", wls_Energy, EmissionFib);
  fiberProperty->AddConstProperty("WLSTIMECONSTANT", 0.5 * ns);
  fPMMA->SetMaterialPropertiesTable(fiberProperty);

  std::vector<G4double> RefractiveIndexClad1 = { 1.49, 1.49, 1.49, 1.49 };
  G4MaterialPropertiesTable* clad1Property   = new G4MaterialPropertiesTable();
  clad1Property->AddProperty("RINDEX", wls_Energy, RefractiveIndexClad1);
  clad1Property->AddProperty("ABSLENGTH", wls_Energy, AbsFiber);
  fPethylene1->SetMaterialPropertiesTable(clad1Property);

  std::vector<G4double> RefractiveIndexClad2 = { 1.42, 1.42, 1.42, 1.42 };
  G4MaterialPropertiesTable* clad2Property   = new G4MaterialPropertiesTable();
  clad2Property->AddProperty("RINDEX", wls_Energy, RefractiveIndexClad2);
  clad2Property->AddProperty("ABSLENGTH", wls_Energy, AbsFiber);
  fPethylene2->SetMaterialPropertiesTable(clad2Property);
}

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

G4VPhysicalVolume* LXeDetectorConstruction::Construct()
{
  // The experimental hall walls are all 1m away from housing walls
  G4double expHall_x = fScint_x + fD_mtl + 1. * m;
  G4double expHall_y = fScint_y + fD_mtl + 1. * m;
  G4double expHall_z = fScint_z + fD_mtl + 1. * m;

  // Create experimental hall
  fExperimentalHall_box =
    new G4Box("expHall_box", expHall_x, expHall_y, expHall_z);
  fExperimentalHall_log =
```

```
    new G4LogicalVolume(fExperimentalHall_box, fVacuum, "expHall_log", 0, 0, 0);
  fExperimentalHall_phys = new G4PVPlacement(
    0, G4ThreeVector(), fExperimentalHall_log, "expHall", 0, false, 0);

  fExperimentalHall_log->SetVisAttributes(G4VisAttributes::GetInvisible());

  // Place the main volume
  if(fMainVolumeOn)
  {
    fMainVolume = new LXeMainVolume(0, G4ThreeVector(), fExperimentalHall_log,
                                    false, 0, this);
  }

  // Place the WLS slab
  if(fWLSslab)
  {
    G4VPhysicalVolume* slab = new LXeWLSSlab(
      0, G4ThreeVector(0., 0., -fScint_z / 2. - fSlab_z - 1. * cm),
      fExperimentalHall_log, false, 0, this);

    // Surface properties for the WLS slab
    G4OpticalSurface* scintWrap = new G4OpticalSurface("ScintWrap");

    new G4LogicalBorderSurface("ScintWrap", slab, fExperimentalHall_phys,
                               scintWrap);

    scintWrap->SetType(dielectric_metal);
    scintWrap->SetFinish(polished);
    scintWrap->SetModel(glisur);

    std::vector<G4double> pp           = { 2.0 * eV, 3.5 * eV };
    std::vector<G4double> reflectivity = { 1.0, 1.0 };
    std::vector<G4double> efficiency   = { 0.0, 0.0 };

    G4MaterialPropertiesTable* scintWrapProperty =
      new G4MaterialPropertiesTable();

    scintWrapProperty->AddProperty("REFLECTIVITY", pp, reflectivity);
    scintWrapProperty->AddProperty("EFFICIENCY", pp, efficiency);
    scintWrap->SetMaterialPropertiesTable(scintWrapProperty);
  }

  //Superficie optica
  G4OpticalSurface* opAirSurface = new G4OpticalSurface("AirSurface");
```

```
  opAirSurface->SetType(dielectric_dielectric);
  opAirSurface->SetFinish(polished);
  opAirSurface->SetModel(unified);

  G4LogicalSkinSurface* airSurface =
    new G4LogicalSkinSurface("AirSurface", fExperimentalHall_log, opAirSurface);

  G4OpticalSurface* opticalSurface = dynamic_cast<G4OpticalSurface*>(
    airSurface->GetSurface(fExperimentalHall_log)->GetSurfaceProperty());
  if(opticalSurface)
    opticalSurface->DumpInfo();

  return fExperimentalHall_phys;
}

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

void LXeDetectorConstruction::ConstructSDandField()
{
  if(!fMainVolume)
    return;

  // PMT SD

  LXePMTSD* pmt = fPmt_SD.Get();
  if(!pmt)
  {
    // Created here so it exists as pmts are being placed
    G4cout << "Construction /LXeDet/pmtSD" << G4endl;
    LXePMTSD* pmt_SD = new LXePMTSD("/LXeDet/pmtSD");
    fPmt_SD.Put(pmt_SD);

    pmt_SD->InitPMTs();
    pmt_SD->SetPmtPositions(fMainVolume->GetPmtPositions());
  }
  else
  {
    pmt->InitPMTs();
    pmt->SetPmtPositions(fMainVolume->GetPmtPositions());
  }
  G4SDManager::GetSDMpointer()->AddNewDetector(fPmt_SD.Get());

  SetSensitiveDetector(fMainVolume->GetLogPhotoCath(), fPmt_SD.Get());
```

```
  // Scint SD

  if(!fScint_SD.Get())
  {
    G4cout << "Construction /LXeDet/scintSD" << G4endl;
    LXeScintSD* scint_SD = new LXeScintSD("/LXeDet/scintSD");
    fScint_SD.Put(scint_SD);
  }
  G4SDManager::GetSDMpointer()->AddNewDetector(fScint_SD.Get());
  SetSensitiveDetector(fMainVolume->GetLogScint(), fScint_SD.Get());
}

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

void LXeDetectorConstruction::SetDimensions(G4ThreeVector dims)
{
  fScint_x = dims[0];
  fScint_y = dims[1];
  fScint_z = dims[2];
  G4RunManager::GetRunManager()->ReinitializeGeometry();
}

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

void LXeDetectorConstruction::SetHousingThickness(G4double d_mtl)
{
  fD_mtl = d_mtl;
  G4RunManager::GetRunManager()->ReinitializeGeometry();
}

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

void LXeDetectorConstruction::SetNX(G4int nx)
{
  fNx = nx;
  G4RunManager::GetRunManager()->ReinitializeGeometry();
}

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo.
```

```
void LXeDetectorConstruction::SetNY(G4int ny)
{
  fNy = ny;
  G4RunManager::GetRunManager()->ReinitializeGeometry();
}

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

void LXeDetectorConstruction::SetNZ(G4int nz)
{
  fNz = nz;
  G4RunManager::GetRunManager()->ReinitializeGeometry();
}

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

void LXeDetectorConstruction::SetPMTRadius(G4double outerRadius_pmt)
{
  fOuterRadius_pmt = outerRadius_pmt;
  G4RunManager::GetRunManager()->ReinitializeGeometry();
}

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

void LXeDetectorConstruction::SetDefaults()
{
  // Resets to default values
  fD_mtl = 0.0635 * cm;

  fScint_x = 24 * cm;
  fScint_y = 24 * cm;
  fScint_z = 1 * cm;

  fNx = 1;       //aqui se controna el numero de PMT
  fNy = 1;
  fNz = 1;

  fOuterRadius_pmt = 0.6 * cm;
  fSphereOn = false;            //quitamos la esfera de true ----> false
  fRefl     = 1.0;

  fNfibers      = 15;
  fWLSslab      = false;
  fMainVolumeOn = true;
  fMainVolume   = nullptr;
```

```
  fSlab_z        = 2.5 * mm;

  G4UImanager::GetUIpointer()->ApplyCommand(
    "/LXe/detector/scintYieldFactor 1.");

  if(fLXe_mt)
    fLXe_mt->AddConstProperty("SCINTILLATIONYIELD", 12000. / MeV);
  if(fMPTPStyrene)
    fMPTPStyrene->AddConstProperty("SCINTILLATIONYIELD", 10. / keV);
}

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

void LXeDetectorConstruction::SetSphereOn(G4bool b)
{
  fSphereOn = b;
  G4RunManager::GetRunManager()->ReinitializeGeometry();
}

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

void LXeDetectorConstruction::SetHousingReflectivity(G4double r)
{
  fRefl = r;
  G4RunManager::GetRunManager()->ReinitializeGeometry();
}

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

void LXeDetectorConstruction::SetWLSSlabOn(G4bool b)
{
  fWLSslab = b;
  G4RunManager::GetRunManager()->ReinitializeGeometry();
}

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

void LXeDetectorConstruction::SetMainVolumeOn(G4bool b)
{
  fMainVolumeOn = b;
  G4RunManager::GetRunManager()->ReinitializeGeometry();
}
```

```
//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo.

void LXeDetectorConstruction::SetNFibers(G4int n)
{
  fNfibers = n;
  G4RunManager::GetRunManager()->ReinitializeGeometry();
}

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

void LXeDetectorConstruction::SetMainScintYield(G4double y)
{
  fLXe_mt->AddConstProperty("SCINTILLATIONYIELD", y / MeV);
}

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

void LXeDetectorConstruction::SetWLSScintYield(G4double y)
{
  fMPTPStyrene->AddConstProperty("SCINTILLATIONYIELD", y / MeV);
}

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

void LXeDetectorConstruction::SetSaveThreshold(G4int save)
{
  fSaveThreshold = save;
  G4RunManager::GetRunManager()->SetRandomNumberStore(true);
}
```

LXeDetectorMessenger.cc

```
#include "LXeDetectorMessenger.hh"

#include "LXeDetectorConstruction.hh"

#include "G4RunManager.hh"
#include "G4Scintillation.hh"
#include "G4UIcmdWithABool.hh"
#include "G4UIcmdWithADouble.hh"
#include "G4UIcmdWithADoubleAndUnit.hh"
#include "G4UIcmdWithAnInteger.hh"
#include "G4UIcmdWith3VectorAndUnit.hh"
#include "G4UIcommand.hh"
#include "G4UIdirectory.hh"

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

LXeDetectorMessenger::LXeDetectorMessenger(LXeDetectorConstruction* detector)
  : fLXeDetector(detector)
{
  // Setup a command directory for detector controls with guidance
  fDetectorDir = new G4UIdirectory("/LXe/detector/");
  fDetectorDir->SetGuidance("Detector geometry control");

  fVolumesDir = new G4UIdirectory("/LXe/detector/volumes/");
  fVolumesDir->SetGuidance("Enable/disable volumes");

  // Various commands for modifying detector geometry
  fDimensionsCmd =
    new G4UIcmdWith3VectorAndUnit("/LXe/detector/dimensions", this);
  fDimensionsCmd->SetGuidance("Set the dimensions of the detector volume.");
  fDimensionsCmd->SetParameterName("scint_x", "scint_y", "scint_z", false);
  fDimensionsCmd->SetDefaultUnit("cm");
  fDimensionsCmd->AvailableForStates(G4State_PreInit, G4State_Idle);
  fDimensionsCmd->SetToBeBroadcasted(false);

  fHousingThicknessCmd =
    new G4UIcmdWithADoubleAndUnit("/LXe/detector/housingThickness", this);
  fHousingThicknessCmd->SetGuidance("Set the thickness of the housing.");
  fHousingThicknessCmd->SetParameterName("d_mtl", false);
  fHousingThicknessCmd->SetDefaultUnit("cm");
  fHousingThicknessCmd->AvailableForStates(G4State_PreInit, G4State_Idle);
  fHousingThicknessCmd->SetToBeBroadcasted(false);
```

```
fPmtRadiusCmd =
  new G4UIcmdWithADoubleAndUnit("/LXe/detector/pmtRadius", this);
fPmtRadiusCmd->SetGuidance("Set the radius of the PMTs.");
fPmtRadiusCmd->SetParameterName("radius", false);
fPmtRadiusCmd->SetDefaultUnit("cm");
fPmtRadiusCmd->AvailableForStates(G4State_PreInit, G4State_Idle);
fPmtRadiusCmd->SetToBeBroadcasted(false);

fNxCmd = new G4UIcmdWithAnInteger("/LXe/detector/nx", this);
fNxCmd->SetGuidance("Set the number of PMTs along the x-dimension.");
fNxCmd->SetParameterName("nx", false);
fNxCmd->AvailableForStates(G4State_PreInit, G4State_Idle);
fNxCmd->SetToBeBroadcasted(false);

fNyCmd = new G4UIcmdWithAnInteger("/LXe/detector/ny", this);
fNyCmd->SetGuidance("Set the number of PMTs along the y-dimension.");
fNyCmd->SetParameterName("ny", false);
fNyCmd->AvailableForStates(G4State_PreInit, G4State_Idle);
fNyCmd->SetToBeBroadcasted(false);

fNzCmd = new G4UIcmdWithAnInteger("/LXe/detector/nz", this);
fNzCmd->SetGuidance("Set the number of PMTs along the z-dimension.");
fNzCmd->SetParameterName("nz", false);
fNzCmd->AvailableForStates(G4State_PreInit, G4State_Idle);
fNzCmd->SetToBeBroadcasted(false);

fSphereCmd = new G4UIcmdWithABool("/LXe/detector/volumes/sphere", this);
fSphereCmd->SetGuidance("Enable/Disable the sphere.");
fSphereCmd->AvailableForStates(G4State_PreInit, G4State_Idle);
fSphereCmd->SetToBeBroadcasted(false);

fReflectivityCmd = new G4UIcmdWithADouble("/LXe/detector/reflectivity", this);
fReflectivityCmd->SetGuidance("Set the reflectivity of the housing.");
fReflectivityCmd->AvailableForStates(G4State_PreInit, G4State_Idle);
fReflectivityCmd->SetToBeBroadcasted(false);

fWlsCmd = new G4UIcmdWithABool("/LXe/detector/volumes/wls", this);
fWlsCmd->SetGuidance("Enable/Disable the WLS slab");
fWlsCmd->AvailableForStates(G4State_PreInit, G4State_Idle);
fWlsCmd->SetToBeBroadcasted(false);

fLxeCmd = new G4UIcmdWithABool("/LXe/detector/volumes/lxe", this);
fLxeCmd->SetGuidance("Enable/Disable the main detector volume.");
fLxeCmd->AvailableForStates(G4State_PreInit, G4State_Idle);
```

```
  fLxeCmd->SetToBeBroadcasted(false);

  fNFibersCmd = new G4UIcmdWithAnInteger("/LXe/detector/nfibers", this);
  fNFibersCmd->SetGuidance("Set the number of WLS fibers in the WLS slab.");
  fNFibersCmd->AvailableForStates(G4State_PreInit, G4State_Idle);
  fNFibersCmd->SetToBeBroadcasted(false);

  fMainScintYield =
    new G4UIcmdWithADouble("/LXe/detector/MainScintYield", this);
  fMainScintYield->SetGuidance("Set scinitillation yield of main volume.");
  fMainScintYield->SetGuidance("Specified in photons/MeV");
  fMainScintYield->AvailableForStates(G4State_PreInit, G4State_Idle);
  fMainScintYield->SetToBeBroadcasted(false);

  fWLSScintYield = new G4UIcmdWithADouble("/LXe/detector/WLSScintYield", this);
  fWLSScintYield->SetGuidance("Set scintillation yield of WLS Slab");
  fWLSScintYield->SetGuidance("Specified in photons/MeV");
  fWLSScintYield->AvailableForStates(G4State_PreInit, G4State_Idle);
  fWLSScintYield->SetToBeBroadcasted(false);

  fSaveThresholdCmd = new G4UIcmdWithAnInteger("/LXe/saveThreshold", this);
  fSaveThresholdCmd->SetGuidance(
    "Set the photon count threshold for saving the random number seed");
  fSaveThresholdCmd->SetParameterName("photons", true);
  fSaveThresholdCmd->SetDefaultValue(4500);
  fSaveThresholdCmd->AvailableForStates(G4State_PreInit, G4State_Idle);

  fDefaultsCmd = new G4UIcommand("/LXe/detector/defaults", this);
  fDefaultsCmd->SetGuidance("Set all detector geometry values to defaults.");
  fDefaultsCmd->AvailableForStates(G4State_PreInit, G4State_Idle);
  fDefaultsCmd->SetToBeBroadcasted(false);
}

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

LXeDetectorMessenger::~LXeDetectorMessenger()
{
  delete fDimensionsCmd;
  delete fHousingThicknessCmd;
  delete fPmtRadiusCmd;
  delete fNxCmd;
  delete fNyCmd;
  delete fNzCmd;
  delete fSphereCmd;
```

```
  delete fWlsCmd;
  delete fLxeCmd;
  delete fNFibersCmd;
  delete fReflectivityCmd;
  delete fMainScintYield;
  delete fWLSScintYield;
  delete fSaveThresholdCmd;
  delete fDefaultsCmd;
  delete fDetectorDir;
  delete fVolumesDir;
}

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

void LXeDetectorMessenger::SetNewValue(G4UIcommand* command, G4String newValue)
{
  if(command == fDimensionsCmd)
  {
    fLXeDetector->SetDimensions(fDimensionsCmd->GetNew3VectorValue(newValue));
  }
  else if(command == fHousingThicknessCmd)
  {
    fLXeDetector->SetHousingThickness(
      fHousingThicknessCmd->GetNewDoubleValue(newValue));
  }
  else if(command == fPmtRadiusCmd)
  {
    fLXeDetector->SetPMTRadius(fPmtRadiusCmd->GetNewDoubleValue(newValue));
  }
  else if(command == fNxCmd)
  {
    fLXeDetector->SetNX(fNxCmd->GetNewIntValue(newValue));
  }
  else if(command == fNyCmd)
  {
    fLXeDetector->SetNY(fNyCmd->GetNewIntValue(newValue));
  }
  else if(command == fNzCmd)
  {
    fLXeDetector->SetNZ(fNzCmd->GetNewIntValue(newValue));
  }
  else if(command == fSphereCmd)
  {
```

```
    fLXeDetector->SetSphereOn(fSphereCmd->GetNewBoolValue(newValue));
  }
  else if(command == fReflectivityCmd)
  {
    fLXeDetector->SetHousingReflectivity(
      fReflectivityCmd->GetNewDoubleValue(newValue));
  }
  else if(command == fWlsCmd)
  {
    fLXeDetector->SetWLSSlabOn(fWlsCmd->GetNewBoolValue(newValue));
  }
  else if(command == fLxeCmd)
  {
    fLXeDetector->SetMainVolumeOn(fLxeCmd->GetNewBoolValue(newValue));
  }
  else if(command == fNFibersCmd)
  {
    fLXeDetector->SetNFibers(fNFibersCmd->GetNewIntValue(newValue));
  }
  else if(command == fMainScintYield)
  {
    fLXeDetector->SetMainScintYield(
      fMainScintYield->GetNewDoubleValue(newValue));
  }
  else if(command == fWLSScintYield)
  {
    fLXeDetector->SetWLSScintYield(fWLSScintYield->GetNewDoubleValue(newValue));
  }
  else if(command == fSaveThresholdCmd)
  {
    fLXeDetector->SetSaveThreshold(fSaveThresholdCmd->GetNewIntValue(newValue));
  }
  else if(command == fDefaultsCmd)
  {
    fLXeDetector->SetDefaults();
    G4RunManager::GetRunManager()->ReinitializeGeometry();
  }
}
```

LXeEventAction.cc

```
#include "LXeEventAction.hh"

#include "LXeDetectorConstruction.hh"
#include "LXeHistoManager.hh"
#include "LXePMTHit.hh"
#include "LXeRun.hh"
#include "LXeScintHit.hh"
#include "LXeTrajectory.hh"

#include "G4Event.hh"
#include "G4EventManager.hh"
#include "G4ios.hh"
#include "G4RunManager.hh"
#include "G4SDManager.hh"
#include "G4SystemOfUnits.hh"
#include "G4Trajectory.hh"
#include "G4TrajectoryContainer.hh"
#include "G4UImanager.hh"
#include "G4VVisManager.hh"

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

LXeEventAction::LXeEventAction(const LXeDetectorConstruction* det)
  : fDetector(det)
  , fScintCollID(-1)
  , fPMTCollID(-1)
  , fVerbose(0)
  , fPMTThreshold(1)
  , fForcedrawphotons(false)
  , fForcenophotons(false)
{
  fEventMessenger = new LXeEventMessenger(this);

  fHitCount                = 0;
  fPhotonCount_Scint       = 0;
  fPhotonCount_Ceren       = 0;
  fAbsorptionCount         = 0;
  fBoundaryAbsorptionCount = 0;
  fTotE                    = 0.0;
    fTotE3                    = 0.0;

  fConvPosSet = false;
  fEdepMax    = 0.0;
```

```
    fEdepMax2    = 0.0;

  fPMTsAboveThreshold = 0;
}

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

LXeEventAction::~LXeEventAction() { delete fEventMessenger; }

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

void LXeEventAction::BeginOfEventAction(const G4Event*)
{
  fHitCount                 = 0;
  fPhotonCount_Scint        = 0;
  fPhotonCount_Ceren        = 0;
  fAbsorptionCount          = 0;
  fBoundaryAbsorptionCount  = 0;
  fTotE                     = 0.0;
    fTotE3                     = 0.0;

  fConvPosSet = false;
  fEdepMax    = 0.0;
    fEdepMax2    = 0.0;

  fPMTsAboveThreshold = 0;

  G4SDManager* SDman = G4SDManager::GetSDMpointer();
  if(fScintCollID < 0)
    fScintCollID = SDman->GetCollectionID("scintCollection");
  if(fPMTCollID < 0)
    fPMTCollID = SDman->GetCollectionID("pmtHitCollection");
}

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

void LXeEventAction::EndOfEventAction(const G4Event* anEvent)
{
  G4TrajectoryContainer* trajectoryContainer =
    anEvent->GetTrajectoryContainer();

  G4int n_trajectories = 0;
  if(trajectoryContainer)
```

```
    n_trajectories = trajectoryContainer->entries();

  // extract the trajectories and draw them
  if(G4VVisManager::GetConcreteInstance())
  {
    for(G4int i = 0; i < n_trajectories; ++i)
    {
      LXeTrajectory* trj =
        (LXeTrajectory*) ((*(anEvent->GetTrajectoryContainer()))[i]);
      if(trj->GetParticleName() == "opticalphoton")
      {
        trj->SetForceDrawTrajectory(fForcedrawphotons);
        trj->SetForceNoDrawTrajectory(fForcenophotons);
      }
      trj->DrawTrajectory();
    }
  }

  LXeScintHitsCollection* scintHC = nullptr;
  LXePMTHitsCollection* pmtHC     = nullptr;
  G4HCofThisEvent* hitsCE         = anEvent->GetHCofThisEvent();

  // Get the hit collections
  if(hitsCE)
  {
    if(fScintCollID >= 0)
    {
      scintHC = (LXeScintHitsCollection*) (hitsCE->GetHC(fScintCollID));
    }
    if(fPMTCollID >= 0)
    {
      pmtHC = (LXePMTHitsCollection*) (hitsCE->GetHC(fPMTCollID));
    }
  }

  // Hits in scintillator
  if(scintHC)
  {
    size_t n_hit = scintHC->entries();
    G4ThreeVector eWeightPos(0.);
    G4double edep;
    G4double edepMax = 0;

    for(size_t i = 0; i < n_hit; ++i)
```

```
    {  // gather info on hits in scintillator
      edep = (*scintHC)[i]->GetEdep();
      fTotE += edep;
      eWeightPos +=
        (*scintHC)[i]->GetPos() * edep;  // calculate energy weighted pos
      if(edep > edepMax)
      {
        edepMax = edep;  // store max energy deposit
        G4ThreeVector posMax = (*scintHC)[i]->GetPos();
        fPosMax              = posMax;
        fEdepMax             = edep;
      }
    }

    G4AnalysisManager::Instance()->FillH1(7, fTotE);

    if(fTotE == 0.)
    {
      if(fVerbose > 0)
        G4cout << "No hits in the scintillator this event." << G4endl;
    }
    else
    {
      // Finish calculation of energy weighted position
      eWeightPos /= fTotE;
      fEWeightPos = eWeightPos;
      if(fVerbose > 0)
      {
        G4cout << "\tEnergy weighted position of hits in LXe : "
               << eWeightPos / mm << G4endl;
      }
    }
    if(fVerbose > 0)
    {
      G4cout << "\tTotal energy deposition in scintillator : " << fTotE / keV
             << " (keV)" << G4endl;
    }
  }

  if(pmtHC)
  {
    G4ThreeVector reconPos(0., 0., 0.);
    size_t pmts = pmtHC->entries();
```

```
    G4ThreeVector eWeightPos2(0.);
    G4double edep2;
    G4double edepMax2 = 0;
    // Gather info from all PMTs
    for(size_t i = 0; i < pmts; ++i)
    {
      fHitCount += (*pmtHC)[i]->GetPhotonCount();
      reconPos += (*pmtHC)[i]->GetPMTPos() * (*pmtHC)[i]->GetPhotonCount();
      if((*pmtHC)[i]->GetPhotonCount() >= fPMTThreshold)
      {
        ++fPMTsAboveThreshold;
      }
      else
      {  // wasn't above the threshold, turn it back off
        (*pmtHC)[i]->SetDrawit(false);
      }


    G4AnalysisManager::Instance()->FillH1(1, fHitCount);
    G4AnalysisManager::Instance()->FillH1(2, fPMTsAboveThreshold);

    if(fHitCount > 0)
    {  // don't bother unless there were hits
      reconPos /= fHitCount;
      if(fVerbose > 0)
      {
        G4cout << "\tReconstructed position of hits in LXe : " << reconPos / mm
               << G4endl;
      }
      fReconPos = reconPos;
    }
    pmtHC->DrawAllHits();
  }

  G4AnalysisManager::Instance()->FillH1(3, fPhotonCount_Scint);
  G4AnalysisManager::Instance()->FillH1(4, fPhotonCount_Ceren);
  G4AnalysisManager::Instance()->FillH1(5, fAbsorptionCount);
  G4AnalysisManager::Instance()->FillH1(6, fBoundaryAbsorptionCount);

    if(pmtHC)
  {
    size_t n_hit = pmtHC->entries();
```

```
    G4ThreeVector eWeightPos(0., 0., 0.);
    G4double edep2;
    G4double edepMax2 = 0;

    for(size_t i = 0; i < n_hit; ++i)
    {  // gather info on hits in scintillator
      edep2 = (*pmtHC)[i]->GetEdep2();
      fTotE3 += edep2;
      eWeightPos +=
        (*pmtHC)[i]->GetPMTPos() * edep2;  // calculate energy weighted pos
      if(edep2 > edepMax2)
      {
        edepMax2 = edep2;  // store max energy deposit
        G4ThreeVector posMax = (*pmtHC)[i]->GetPMTPos();
        fPosMax              = posMax;
        fEdepMax2             = edep2;
      }

    G4AnalysisManager::Instance()->FillH1(8, fTotE3);}

   if(fTotE3 == 0.)
    {
      if(fVerbose > 0)
        G4cout << "No hits in the scintillator this event." << G4endl;
    }
    else
    {
      // Finish calculation of energy weighted position
      eWeightPos /= fTotE3;
      fEWeightPos = eWeightPos;
      if(fVerbose > 0)
      {
        G4cout << "\tEnergy weighted position of hits in LXe : "
               << eWeightPos / mm << G4endl;
      }
    }
    if(fVerbose > 0)
    {
      G4cout << "\tTotal energy deposition in PMT: " << fTotE3 / keV
             << " (keV)" << G4endl;


    }
```

```
  }

  if(fVerbose > 0)
  {
    // End of event output. later to be controlled by a verbose level
    G4cout << "\tNumber of photons that hit PMTs in this event : " << fHitCount
           << G4endl;
    G4cout << "\tNumber of PMTs above threshold(" << fPMTThreshold
           << ") : " << fPMTsAboveThreshold << G4endl;
    G4cout << "\tNumber of photons produced by scintillation in this event : "
           << fPhotonCount_Scint << G4endl;
    G4cout << "\tNumber of photons produced by cerenkov in this event : "
           << fPhotonCount_Ceren << G4endl;
    G4cout << "\tNumber of photons absorbed (OpAbsorption) in this event : "
           << fAbsorptionCount << G4endl;
    G4cout << "\tNumber of photons absorbed at boundaries (OpBoundary) in "
           << "this event : " << fBoundaryAbsorptionCount << G4endl;
    G4cout << "Unaccounted for photons in this event : "
           << (fPhotonCount_Scint + fPhotonCount_Ceren - fAbsorptionCount -
               fHitCount - fBoundaryAbsorptionCount)
           << G4endl;
  }

  // update the run statistics
  LXeRun* run = static_cast<LXeRun*>(
    G4RunManager::GetRunManager()->GetNonConstCurrentRun());

  run->IncHitCount(fHitCount);
  run->IncPhotonCount_Scint(fPhotonCount_Scint);
  run->IncPhotonCount_Ceren(fPhotonCount_Ceren);
  run->IncEDep(fTotE);
   run->IncEDep2(fTotE3);
  run->IncAbsorption(fAbsorptionCount);
  run->IncBoundaryAbsorption(fBoundaryAbsorptionCount);
  run->IncHitsAboveThreshold(fPMTsAboveThreshold);

  // If we have set the flag to save 'special' events, save here
  if(fPhotonCount_Scint + fPhotonCount_Ceren < fDetector->GetSaveThreshold())
  {
    G4RunManager::GetRunManager()->rndmSaveThisEvent();
  }
}
```

LXeEventMessenger.cc

```
#include "LXeEventMessenger.hh"

#include "LXeEventAction.hh"

#include "G4UIcmdWithABool.hh"
#include "G4UIcmdWithAnInteger.hh"

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

LXeEventMessenger::LXeEventMessenger(LXeEventAction* event)
  : fLXeEvent(event)
{
  fVerboseCmd = new G4UIcmdWithAnInteger("/LXe/eventVerbose", this);
  fVerboseCmd->SetGuidance("Set the verbosity of event data.");
  fVerboseCmd->SetParameterName("verbose", true);
  fVerboseCmd->SetDefaultValue(1);

  fPmtThresholdCmd = new G4UIcmdWithAnInteger("/LXe/pmtThreshold", this);
  fPmtThresholdCmd->SetGuidance("Set the pmtThreshold (in # of photons)");

  fForceDrawPhotonsCmd = new G4UIcmdWithABool("/LXe/forceDrawPhotons", this);
  fForceDrawPhotonsCmd->SetGuidance("Force drawing of photons.");
  fForceDrawPhotonsCmd->SetGuidance(
    "(Higher priority than /LXe/forceDrawNoPhotons)");

  fForceDrawNoPhotonsCmd =
    new G4UIcmdWithABool("/LXe/forceDrawNoPhotons", this);
  fForceDrawNoPhotonsCmd->SetGuidance("Force no drawing of photons.");
  fForceDrawNoPhotonsCmd->SetGuidance(
    "(Lower priority than /LXe/forceDrawPhotons)");
}

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

LXeEventMessenger::~LXeEventMessenger()
{
  delete fVerboseCmd;
  delete fPmtThresholdCmd;
  delete fForceDrawPhotonsCmd;
  delete fForceDrawNoPhotonsCmd;
}
```

```
void LXeEventMessenger::SetNewValue(G4UIcommand* command, G4String newValue)
{
  if(command == fVerboseCmd)
  {
    fLXeEvent->SetEventVerbose(fVerboseCmd->GetNewIntValue(newValue));
  }
  else if(command == fPmtThresholdCmd)
  {
    fLXeEvent->SetPMTThreshold(fPmtThresholdCmd->GetNewIntValue(newValue));
  }
  else if(command == fForceDrawPhotonsCmd)
  {
    fLXeEvent->SetForceDrawPhotons(
      fForceDrawPhotonsCmd->GetNewBoolValue(newValue));
  }
  else if(command == fForceDrawNoPhotonsCmd)
  {
    fLXeEvent->SetForceDrawNoPhotons(
      fForceDrawNoPhotonsCmd->GetNewBoolValue(newValue));
  }
}
```

LXeHistoManager.cc

```
#include "LXeHistoManager.hh"

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

LXeHistoManager::LXeHistoManager()
  : fFileName("lxe")
{
  Book();
}

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

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

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

void LXeHistoManager::Book()
{
  G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();
  analysisManager->SetFileName(fFileName);
  analysisManager->SetVerboseLevel(1);
  analysisManager->SetActivation(true);
  G4int nbins   = 100;
  G4double vmin = 0.;
  G4double vmax = 100.;

  // 0
  analysisManager->CreateH1("0", "dummy", nbins, vmin, vmax);
  // 1
  analysisManager->CreateH1("hits per event", "hits per event", nbins, vmin,
                            vmax);
  // 2
  analysisManager->CreateH1("hits above threshold",
                            "hits per event above threshold", nbins, vmin,
                            vmax);
  // 3
  analysisManager->CreateH1("scintillation", "scintillation photons per event",
                            nbins, vmin, vmax);
  // 4
  analysisManager->CreateH1("Cerenkov", "Cerenkov photons per event", nbins,
                            vmin, vmax);
```

```
  // 5
  analysisManager->CreateH1("absorbed", "absorbed photons per event", nbins,
                            vmin, vmax);
  // 6
  analysisManager->CreateH1("boundary absorbed",
                            "photons absorbed at boundary per event", nbins,
                            vmin, vmax);
  // 7
  analysisManager->CreateH1(
    "E dep", "energy deposition in scintillator per event", nbins, vmin, vmax);

  // 8
  analysisManager->CreateH1(
    "E dep2", "energy deposition in PMT per event", nbins, vmin, vmax);

  // Create all histograms as inactivated
  for(G4int i = 0; i < analysisManager->GetNofH1s(); ++i)
  {
    analysisManager->SetH1Activation(i, false);
  }
}
```

LXeMainVolume.cc

```
#include "LXeMainVolume.hh"

#include "globals.hh"
#include "G4Box.hh"
#include "G4Colour.hh"
#include "G4LogicalSkinSurface.hh"
#include "G4LogicalBorderSurface.hh"
#include "G4LogicalVolume.hh"
#include "G4Material.hh"
#include "G4MaterialPropertiesTable.hh"
#include "G4OpticalSurface.hh"
#include "G4Sphere.hh"
#include "G4SystemOfUnits.hh"
#include "G4Tubs.hh"
#include "G4VisAttributes.hh"

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......

LXeMainVolume::LXeMainVolume(G4RotationMatrix* pRot, const G4ThreeVector& tlate,
                             G4LogicalVolume* pMotherLogical, G4bool pMany,
                             G4int pCopyNo, LXeDetectorConstruction* c)
  // Pass info to the G4PVPlacement constructor
  : G4PVPlacement(pRot, tlate,
                  // Temp logical volume must be created here
                  new G4LogicalVolume(new G4Box("temp", 1, 1, 1),
                                      G4Material::GetMaterial("Air"), "temp",
                                      0, 0, 0),
                  "housing", pMotherLogical, pMany, pCopyNo)
  , fConstructor(c)
{
  CopyValues();

  G4double housing_x = fScint_x + 2. * fD_mtl;
  G4double housing_y = fScint_y + 2. * fD_mtl;
  G4double housing_z = fScint_z + 2. * fD_mtl;

  //************************** housing and scintillator
  fScint_box =
    new G4Box("scint_box", fScint_x / 2., fScint_y / 2., fScint_z / 2.);
  fHousing_box =
    new G4Box("housing_box", housing_x / 2., housing_y / 2., housing_z / 2.);

  fScint_log   = new G4LogicalVolume(fScint_box,
      G4Material::GetMaterial("G4_PLASTIC_SC_VINYLTOLUENE"),"scint_log", 0, 0, 0);
```

```
fHousing_log = new G4LogicalVolume(
  fHousing_box, G4Material::GetMaterial("Air"), "housing_log", 0, 0, 0);

new G4PVPlacement(0, G4ThreeVector(), fScint_log, "scintillator",
                  fHousing_log, false, 0);

//*************** Miscellaneous sphere to demonstrate skin surfaces
fSphere = new G4Sphere("sphere", 0., 2. * cm, 0. * deg, 360. * deg, 0. * deg,
                       360. * deg);
fSphere_log =
  new G4LogicalVolume(fSphere, G4Material::GetMaterial("Al"), "sphere_log");
if(fSphereOn)
  new G4PVPlacement(0, G4ThreeVector(5. * cm, 5. * cm, 5. * cm), fSphere_log,
                    "sphere", fScint_log, false, 0);

//****************** Build PMTs
//G4double innerRadius_pmt   = 0.;
G4double height_pmt        = fD_mtl / 2.;
//G4double startAngle_pmt    = 0.;
//G4double spanningAngle_pmt = 360. * deg;

fPmt = new G4Box("pmt_tube", fOuterRadius_pmt, fOuterRadius_pmt, height_pmt
fPhotocath = new G4Box("photocath_tube", fOuterRadius_pmt, fOuterRadius_pmt,
                       height_pmt / 2.);

fPmt_log =
  new G4LogicalVolume(fPmt, G4Material::GetMaterial("Glass"), "pmt_log");
fPhotocath_log = new G4LogicalVolume(
  fPhotocath, G4Material::GetMaterial("G4_Si"), "photocath_log");

new G4PVPlacement(0, G4ThreeVector(0., 0., -height_pmt / 2.), fPhotocath_log,
                  "photocath", fPmt_log, false, 0);

//***********Arrange pmts around the outside of housing**********

G4double dx = fScint_x / fNx;
G4double dy = fScint_y / fNy;
G4double dz = fScint_z / fNz;

G4double x, y, z;
G4double xmin = -fScint_x / 2. - dx / 2.;
```

```
  G4double ymin = -fScint_y / 2. - dy / 2.;
  G4double zmin = -fScint_z / 2. - dz / 2.;
  G4int k       = 0;

  G4RotationMatrix* rm_z = new G4RotationMatrix();
  rm_z->rotateY(180. * deg);
  z = fScint_z / 2. + height_pmt;  // back
  PlacePMTs(fPmt_log, rm_z, x, y, dx, dy, xmin, ymin, fNx, fNy, x, y, z, k);

  VisAttributes();
  SurfaceProperties();

  SetLogicalVolume(fHousing_log);
}

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

void LXeMainVolume::CopyValues()
{
  fScint_x         = fConstructor->GetScintX();
  fScint_y         = fConstructor->GetScintY();
  fScint_z         = fConstructor->GetScintZ();
  fD_mtl           = fConstructor->GetHousingThickness();
  fNx              = fConstructor->GetNX();
  fNy              = fConstructor->GetNY();
  fNz              = fConstructor->GetNZ();
  fOuterRadius_pmt = fConstructor->GetPMTRadius();
  fSphereOn        = fConstructor->GetSphereOn();
  fRefl            = fConstructor->GetHousingReflectivity();
}

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

void LXeMainVolume::PlacePMTs(G4LogicalVolume* pmt_log, G4RotationMatrix* rot,
                              G4double& a, G4double& b, G4double da,
                              G4double db, G4double amin, G4double bmin,
                              G4int na, G4int nb, G4double& x, G4double& y,
                              G4double& z, G4int& k)
{
  a = amin;
  for(G4int j = 1; j <= na; ++j)
  {
    a += da;
    b = bmin;
```

```
    for(G4int i = 1; i <= nb; ++i)
    {
      b += db;
      new G4PVPlacement(rot, G4ThreeVector(x, y, z), pmt_log, "pmt",
                        fHousing_log, false, k);
      fPmtPositions.push_back(G4ThreeVector(x, y, z));
      ++k;
    }
  }
}

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

void LXeMainVolume::VisAttributes()
{
  G4VisAttributes* housing_va = new G4VisAttributes(G4Colour(0.8, 0.8, 0.8));
  fHousing_log->SetVisAttributes(housing_va);

  G4VisAttributes* sphere_va = new G4VisAttributes();
  sphere_va->SetForceSolid(true);
  fSphere_log->SetVisAttributes(sphere_va);
}

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

void LXeMainVolume::SurfaceProperties()
{
  std::vector<G4double> ephoton = { 7.0 * eV, 7.14 * eV };

  //**Scintillator housing properties
  std::vector<G4double> reflectivity     = { fRefl, fRefl };
  std::vector<G4double> efficiency       = { 0.0, 0.0 };
  G4MaterialPropertiesTable* scintHsngPT = new G4MaterialPropertiesTable();
  scintHsngPT->AddProperty("REFLECTIVITY", ephoton, reflectivity);
  scintHsngPT->AddProperty("EFFICIENCY", ephoton, efficiency);
  G4OpticalSurface* OpScintHousingSurface =
    new G4OpticalSurface("HousingSurface", unified, polished, dielectric_metal);
  OpScintHousingSurface->SetMaterialPropertiesTable(scintHsngPT);

  //**Sphere surface properties
  std::vector<G4double> sphereReflectivity = { 1.0, 1.0 };
  std::vector<G4double> sphereEfficiency   = { 0.0, 0.0 };
  G4MaterialPropertiesTable* spherePT      = new G4MaterialPropertiesTable();
  spherePT->AddProperty("REFLECTIVITY", ephoton, sphereReflectivity);
```

```
  spherePT->AddProperty("EFFICIENCY", ephoton, sphereEfficiency);
  G4OpticalSurface* OpSphereSurface =
    new G4OpticalSurface("SphereSurface", unified, polished, dielectric_metal);
  OpSphereSurface->SetMaterialPropertiesTable(spherePT);

  //**Photocathode surface properties
  std::vector<G4double> photocath_EFF     = { 1., 1. };
  std::vector<G4double> photocath_ReR     = { 1.92, 1.92 };
  std::vector<G4double> photocath_ImR     = { 1.69, 1.69 };
  G4MaterialPropertiesTable* photocath_mt = new G4MaterialPropertiesTable();
  photocath_mt->AddProperty("EFFICIENCY", ephoton, photocath_EFF);
  photocath_mt->AddProperty("REALRINDEX", ephoton, photocath_ReR);
  photocath_mt->AddProperty("IMAGINARYRINDEX", ephoton, photocath_ImR);
  G4OpticalSurface* photocath_opsurf = new G4OpticalSurface(
    "photocath_opsurf", glisur, polished, dielectric_metal);
  photocath_opsurf->SetMaterialPropertiesTable(photocath_mt);

  //**Create logical skin surfaces
  new G4LogicalSkinSurface("photocath_surf", fHousing_log,
                           OpScintHousingSurface);
  new G4LogicalSkinSurface("sphere_surface", fSphere_log, OpSphereSurface);
  new G4LogicalSkinSurface("photocath_surf", fPhotocath_log, photocath_opsurf);
}
```

LXePMTHit.cc

```
#include "LXePMTHit.hh"

#include "G4Colour.hh"
#include "G4ios.hh"
#include "G4LogicalVolume.hh"
#include "G4VisAttributes.hh"
#include "G4VPhysicalVolume.hh"
#include "G4VVisManager.hh"

G4ThreadLocal G4Allocator<LXePMTHit>* LXePMTHitAllocator = nullptr;


LXePMTHit::LXePMTHit()
  : fPmtNumber(-1)
  , fPhotons(0)
    , fEdep2(0.)  //nuevo
  , fPhysVol(nullptr)
  , fDrawit(false)
{}


LXePMTHit::LXePMTHit(G4VPhysicalVolume* pVol)
  : fPhysVol(pVol)
{}


LXePMTHit::~LXePMTHit() {}


LXePMTHit::LXePMTHit(const LXePMTHit& right)
  : G4VHit()
{
  fEdep2    = right.fEdep2;
  fPmtNumber = right.fPmtNumber;
  fPhotons   = right.fPhotons;
  fPhysVol   = right.fPhysVol;
  fDrawit    = right.fDrawit;
}


const LXePMTHit& LXePMTHit::operator=(const LXePMTHit& right)
{

  fEdep2    = right.fEdep2;
```

```
  fPmtNumber = right.fPmtNumber;
  fPhotons   = right.fPhotons;
  fPhysVol   = right.fPhysVol;
  fDrawit    = right.fDrawit;
  return *this;
}


G4bool LXePMTHit::operator==(const LXePMTHit& right) const
{
  return (fPmtNumber == right.fPmtNumber);
}


void LXePMTHit::Draw()
{
  if(fDrawit && fPhysVol)
  {
    G4VVisManager* pVVisManager = G4VVisManager::GetConcreteInstance();
    if(pVVisManager)
    {
      G4VisAttributes attribs(G4Colour(1., 0., 0.));
      attribs.SetForceSolid(true);
      G4RotationMatrix rot;
      if(fPhysVol->GetRotation())
        rot = *(fPhysVol->GetRotation());
      G4Transform3D trans(rot, fPhysVol->GetTranslation());
      pVVisManager->Draw(*fPhysVol, attribs, trans);
    }
  }
}

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

void LXePMTHit::Print() {}
```

LXePMTSD.cc

```
#include "LXePMTSD.hh"

#include "LXeDetectorConstruction.hh"
#include "LXePMTHit.hh"
#include "LXeUserTrackInformation.hh"

#include "G4ios.hh"
#include "G4LogicalVolume.hh"
#include "G4ParticleDefinition.hh"
#include "G4ParticleTypes.hh"
#include "G4SDManager.hh"
#include "G4Step.hh"
#include "G4TouchableHistory.hh"
#include "G4Track.hh"
#include "G4VPhysicalVolume.hh"
#include "G4VTouchable.hh"

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

LXePMTSD::LXePMTSD(G4String name)
  : G4VSensitiveDetector(name)
  , fPMTHitCollection(nullptr)
  , fPMTPositionsX(nullptr)
  , fPMTPositionsY(nullptr)
  , fPMTPositionsZ(nullptr)
  , fHitCID(-1)
{
  collectionName.insert("pmtHitCollection");
}


LXePMTSD::~LXePMTSD()
{
  delete fPMTPositionsX;
  delete fPMTPositionsY;
  delete fPMTPositionsZ;
}


void LXePMTSD::SetPmtPositions(const std::vector<G4ThreeVector>& positions)
{
  for(size_t i = 0; i < positions.size(); ++i)
  {
    if(fPMTPositionsX)
```

```
      fPMTPositionsX->push_back(positions[i].x());
    if(fPMTPositionsY)
      fPMTPositionsY->push_back(positions[i].y());
    if(fPMTPositionsZ)
      fPMTPositionsZ->push_back(positions[i].z());
  }
}


void LXePMTSD::Initialize(G4HCofThisEvent* hitsCE)
{
  fPMTHitCollection =
    new LXePMTHitsCollection(SensitiveDetectorName, collectionName[0]);

  if(fHitCID < 0)
  {
    fHitCID = G4SDManager::GetSDMpointer()->GetCollectionID(fPMTHitCollection);
  }
  hitsCE->AddHitsCollection(fHitCID, fPMTHitCollection);
}

G4bool LXePMTSD::ProcessHits(G4Step* aStep, G4TouchableHistory*)
{
  G4double edep2 = aStep->GetTotalEnergyDeposit();
  if(edep2 == 0.)
    return false;

  G4StepPoint* thePrePoint = aStep->GetPreStepPoint();
  G4TouchableHistory* theTouchable =
    (G4TouchableHistory*) (aStep->GetPreStepPoint()->GetTouchable());
  G4VPhysicalVolume* thePrePV = theTouchable->GetVolume();

  G4StepPoint* thePostPoint = aStep->GetPostStepPoint();

  G4ThreeVector pos2 = thePrePoint->GetPosition() + thePostPoint->GetPosition();
  pos2 /= 2.;

  LXePMTHit* PMTHit = new LXePMTHit(thePrePV);

  PMTHit->SetEdep2(edep2);
  PMTHit->SetPMTPos2(pos2);

  fPMTHitCollection->insert(PMTHit);
  return true;
```

```
}
G4bool LXePMTSD::ProcessHits_boundary(const G4Step* aStep, G4TouchableHistory*)
{
  if(aStep->GetTrack()->GetDefinition() !=
     G4OpticalPhoton::OpticalPhotonDefinition())
    return false;
  G4int pmtNumber =
    aStep->GetPostStepPoint()->GetTouchable()->GetReplicaNumber(1);
  G4VPhysicalVolume* physVol =
    aStep->GetPostStepPoint()->GetTouchable()->GetVolume(1);

  size_t n       = fPMTHitCollection->entries();
  LXePMTHit* hit = nullptr;
  for(size_t i = 0; i < n; ++i)
  {
    if((*fPMTHitCollection)[i]->GetPMTNumber() == pmtNumber)
    {
      hit = (*fPMTHitCollection)[i];
      break;
    }
  }
  if(hit == nullptr)
  {
    hit = new LXePMTHit();
    hit->SetPMTNumber(pmtNumber);
    hit->SetPMTPhysVol(physVol);
    fPMTHitCollection->insert(hit);
    hit->SetPMTPos((*fPMTPositionsX)[pmtNumber], (*fPMTPositionsY)[pmtNumber],
                   (*fPMTPositionsZ)[pmtNumber]);
  }
  hit->IncPhotonCount();
  if(!LXeDetectorConstruction::GetSphereOn())
  {
    hit->SetDrawit(true);
  }
  else
  {
    LXeUserTrackInformation* trackInfo =
      (LXeUserTrackInformation*) aStep->GetTrack()->GetUserInformation();
    if(trackInfo->GetTrackStatus() & hitSphere)
      hit->SetDrawit(true);
  }
  return true;
}
```

LXePrimaryGeneratorAction.cc

```
#include "LXePrimaryGeneratorAction.hh"

#include "globals.hh"
#include "G4Event.hh"
#include "G4ParticleDefinition.hh"
#include "G4ParticleGun.hh"
#include "G4ParticleTable.hh"
#include "G4SystemOfUnits.hh"


LXePrimaryGeneratorAction::LXePrimaryGeneratorAction()
{
  G4int n_particle = 1;
  fParticleGun     = new G4ParticleGun(n_particle);

  G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();

  G4String particleName;
  fParticleGun->SetParticleDefinition(
    particleTable->FindParticle(particleName = "mu+"));
  fParticleGun->SetParticleEnergy(1. * GeV);
  fParticleGun->SetParticlePosition(G4ThreeVector(0., 0., -20. * cm));
  fParticleGun->SetParticleMomentumDirection(G4ThreeVector(0., 0., 1.));
}

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

void LXePrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{
  fParticleGun->GeneratePrimaryVertex(anEvent);
}
```

LXeRun.cc

```
#include "LXeRun.hh"
#include "G4SystemOfUnits.hh"

LXeRun::LXeRun()
  : G4Run()
{
  fHitCount = fHitCount2 = 0;
  fPhotonCount_Scint = fPhotonCount_Scint2 = 0;
  fPhotonCount_Ceren = fPhotonCount_Ceren2 = 0;
  fAbsorptionCount = fAbsorptionCount2 = 0;
  fBoundaryAbsorptionCount = fBoundaryAbsorptionCount2 = 0;
  fPMTsAboveThreshold = fPMTsAboveThreshold2 = 0;

  fTotE = fTotE2 = 0.0;
    fTotE3 = fTotE4 = 0.0;
}

LXeRun::~LXeRun() {}

void LXeRun::Merge(const G4Run* run)
{
  const LXeRun* localRun = static_cast<const LXeRun*>(run);

  fHitCount += localRun->fHitCount;
  fHitCount2 += localRun->fHitCount2;
  fPMTsAboveThreshold += localRun->fPMTsAboveThreshold;
  fPMTsAboveThreshold2 += localRun->fPMTsAboveThreshold2;
  fPhotonCount_Scint += localRun->fPhotonCount_Scint;
  fPhotonCount_Scint2 += localRun->fPhotonCount_Scint2;
  fPhotonCount_Ceren += localRun->fPhotonCount_Ceren;
  fPhotonCount_Ceren2 += localRun->fPhotonCount_Ceren2;
  fAbsorptionCount += localRun->fAbsorptionCount;
  fAbsorptionCount2 += localRun->fAbsorptionCount2;
  fBoundaryAbsorptionCount += localRun->fBoundaryAbsorptionCount;
  fBoundaryAbsorptionCount2 += localRun->fBoundaryAbsorptionCount2;
  fTotE += localRun->fTotE;
  fTotE2 += localRun->fTotE2;
    fTotE3 += localRun->fTotE3;
    fTotE4 += localRun->fTotE4;

  G4Run::Merge(run);
}

void LXeRun::EndOfRun()
```

```
{
  G4cout << "\n ======================== run summary ======================\n";

  G4int prec = G4cout.precision();

  G4double n_evt = (G4double) numberOfEvent;
  G4cout << "The run was " << numberOfEvent << " events." << G4endl;

  G4cout.precision(4);
  G4double hits     = G4double(fHitCount) / n_evt;
  G4double hits2    = G4double(fHitCount2) / n_evt;
  G4double rms_hits = hits2 - hits * hits;
  if(rms_hits > 0.)
    rms_hits = std::sqrt(rms_hits / n_evt);
  else
    rms_hits = 0.;
  G4cout << "Number of hits per event:\t " << hits << " +- " << rms_hits
         << G4endl;

  G4double hitsAbove     = G4double(fPMTsAboveThreshold) / n_evt;
  G4double hitsAbove2    = G4double(fPMTsAboveThreshold2) / n_evt;
  G4double rms_hitsAbove = hitsAbove2 - hitsAbove * hitsAbove;
  if(rms_hitsAbove > 0.)
    rms_hitsAbove = std::sqrt(rms_hitsAbove / n_evt);
  else
    rms_hitsAbove = 0.;

  G4cout << "Number of hits per event above threshold:\t " << hitsAbove
         << " +- " << rms_hitsAbove << G4endl;

  G4double scint     = G4double(fPhotonCount_Scint) / n_evt;
  G4double scint2    = G4double(fPhotonCount_Scint2) / n_evt;
  G4double rms_scint = scint2 - scint * scint;
  if(rms_scint > 0.)
    rms_scint = std::sqrt(rms_scint / n_evt);
  else
    rms_scint = 0.;

  G4cout << "Number of scintillation photons per event :\t " << scint << " +- "
         << rms_scint << G4endl;

  G4double ceren     = G4double(fPhotonCount_Ceren) / n_evt;
  G4double ceren2    = G4double(fPhotonCount_Ceren2) / n_evt;
```

```
if(rms_ceren > 0.)
  rms_ceren = std::sqrt(rms_ceren / n_evt);
else
  rms_ceren = 0.;

G4cout << "Number of Cerenkov photons per event:\t " << ceren << " +- "
       << rms_ceren << G4endl;

G4double absorb     = G4double(fAbsorptionCount) / n_evt;
G4double absorb2    = G4double(fAbsorptionCount2) / n_evt;
G4double rms_absorb = absorb2 - absorb * absorb;
if(rms_absorb > 0.)
  rms_absorb = std::sqrt(rms_absorb / n_evt);
else
  rms_absorb = 0.;

G4cout << "Number of absorbed photons per event :\t " << absorb << " +- "
       << rms_absorb << G4endl;

G4double bdry     = G4double(fBoundaryAbsorptionCount) / n_evt;
G4double bdry2    = G4double(fBoundaryAbsorptionCount2) / n_evt;
G4double rms_bdry = bdry2 - bdry * bdry;
if(rms_bdry > 0.)
  rms_bdry = std::sqrt(rms_bdry / n_evt);
else
  rms_bdry = 0.;

G4cout << "Number of photons absorbed at boundary per event:\t " << bdry
       << " +- " << rms_bdry << G4endl;

G4double en     = fTotE / n_evt;
G4double en2    = fTotE2 / n_evt;
G4double rms_en = en2 - en * en;
if(rms_en > 0.)
  rms_en = std::sqrt(rms_en / n_evt);
else
  rms_en = 0.;

G4cout << "Total energy deposition in scintillator per event:\t " << en / keV
       << " +- " << rms_en / keV << " keV." << G4endl;

       G4double en3     = fTotE3 / n_evt;
       G4double en4    = fTotE4 / n_evt;
```

```
        G4double rms_en2 = en4 - en3 * en3;
        if(rms_en2 > 0.)
        rms_en2 = std::sqrt(rms_en2 / n_evt);
        else
        rms_en2 = 0.;

        G4cout << "Total energy deposition in PMT per event:\t " << en3 / keV
        << " +- " << rms_en2 / keV << " keV." << G4endl;

  G4cout << G4endl;
  G4cout.precision(prec);
}
```

LXeRunAction.cc

```
#include "LXeRunAction.hh"
#include "LXeHistoManager.hh"
#include "LXeRun.hh"

LXeRunAction::LXeRunAction()
  : fRun(nullptr)
  , fHistoManager(nullptr)
{
  fHistoManager = new LXeHistoManager();
}

LXeRunAction::~LXeRunAction() { delete fHistoManager; }

G4Run* LXeRunAction::GenerateRun()
{
  fRun = new LXeRun();
  return fRun;
}

void LXeRunAction::BeginOfRunAction(const G4Run*)
{
  G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();
  if(analysisManager->IsActive())
  {
    analysisManager->OpenFile();
  }
}

void LXeRunAction::EndOfRunAction(const G4Run*)
{
  if(isMaster)
    fRun->EndOfRun();

  G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();
  if(analysisManager->IsActive())
  {
    analysisManager->Write();
    analysisManager->CloseFile();
  }
}
```

LXeScintHit.cc

```
#include "LXeScintHit.hh"

#include "G4ios.hh"
#include "G4Colour.hh"
#include "G4LogicalVolume.hh"
#include "G4VisAttributes.hh"
#include "G4VPhysicalVolume.hh"
#include "G4VVisManager.hh"

G4ThreadLocal G4Allocator<LXeScintHit>* LXeScintHitAllocator = nullptr;

LXeScintHit::LXeScintHit()
  : fEdep(0.)
  , fPos(0.)
  , fPhysVol(nullptr)
{}

LXeScintHit::LXeScintHit(G4VPhysicalVolume* pVol)
  : fPhysVol(pVol)
{}

LXeScintHit::~LXeScintHit() {}

LXeScintHit::LXeScintHit(const LXeScintHit& right)
  : G4VHit()
{
  fEdep    = right.fEdep;
  fPos     = right.fPos;
  fPhysVol = right.fPhysVol;
}

const LXeScintHit& LXeScintHit::operator=(const LXeScintHit& right)
{
  fEdep    = right.fEdep;
  fPos     = right.fPos;
  fPhysVol = right.fPhysVol;
  return *this;
}

G4bool LXeScintHit::operator==(const LXeScintHit&) const
{
  return false;
}
```

LXeScintSD.cc

```
#include "LXeScintSD.hh"

#include "LXeScintHit.hh"

#include "G4ios.hh"
#include "G4LogicalVolume.hh"
#include "G4ParticleDefinition.hh"
#include "G4SDManager.hh"
#include "G4Step.hh"
#include "G4TouchableHistory.hh"
#include "G4Track.hh"
#include "G4VPhysicalVolume.hh"
#include "G4VProcess.hh"
#include "G4VTouchable.hh"

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

LXeScintSD::LXeScintSD(G4String name)
  : G4VSensitiveDetector(name)
  , fHitsCID(-1)
{
  fScintCollection = nullptr;
  collectionName.insert("scintCollection");
}

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

LXeScintSD::~LXeScintSD() {}

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

void LXeScintSD::Initialize(G4HCofThisEvent* hitsCE)
{
  fScintCollection =
    new LXeScintHitsCollection(SensitiveDetectorName, collectionName[0]);

  if(fHitsCID < 0)
  {
    fHitsCID = G4SDManager::GetSDMpointer()->GetCollectionID(fScintCollection);
  }
  hitsCE->AddHitsCollection(fHitsCID, fScintCollection);
}
```

```
//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo.....

G4bool LXeScintSD::ProcessHits(G4Step* aStep, G4TouchableHistory*)
{
  G4double edep = aStep->GetTotalEnergyDeposit();
  if(edep == 0.)
    return false;

  G4StepPoint* thePrePoint = aStep->GetPreStepPoint();
  G4TouchableHistory* theTouchable =
    (G4TouchableHistory*) (aStep->GetPreStepPoint()->GetTouchable());
  G4VPhysicalVolume* thePrePV = theTouchable->GetVolume();

  G4StepPoint* thePostPoint = aStep->GetPostStepPoint();

  // Get the average position of the hit
  G4ThreeVector pos = thePrePoint->GetPosition() + thePostPoint->GetPosition();
  pos /= 2.;

  LXeScintHit* scintHit = new LXeScintHit(thePrePV);

  scintHit->SetEdep(edep);
  scintHit->SetPos(pos);

  fScintCollection->insert(scintHit);

  return true;
}
```

LXeStackingAction.cc

```
#include "LXeStackingAction.hh"

#include "LXeEventAction.hh"

#include "G4OpticalPhoton.hh"
#include "G4Track.hh"
#include "G4VProcess.hh"

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

LXeStackingAction::LXeStackingAction(LXeEventAction* ea)
  : fEventAction(ea)
{}

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

LXeStackingAction::~LXeStackingAction() {}

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

G4ClassificationOfNewTrack LXeStackingAction::ClassifyNewTrack(
  const G4Track* aTrack)
{
  // Count what process generated the optical photons
  if(aTrack->GetDefinition() == G4OpticalPhoton::OpticalPhotonDefinition())
  {
    // particle is optical photon
    if(aTrack->GetParentID() > 0)
    {
      // particle is secondary
      if(aTrack->GetCreatorProcess()->GetProcessName() == "Scintillation")
        fEventAction->IncPhotonCount_Scint();
      else if(aTrack->GetCreatorProcess()->GetProcessName() == "Cerenkov")
        fEventAction->IncPhotonCount_Ceren();
    }
  }
  return fUrgent;
}
```

LXeSteppingAction.cc

```
#include "LXeSteppingAction.hh"

#include "LXeEventAction.hh"
#include "LXePMTSD.hh"
#include "LXeSteppingMessenger.hh"
#include "LXeTrajectory.hh"
#include "LXeUserTrackInformation.hh"

#include "G4OpticalPhoton.hh"
#include "G4ProcessManager.hh"
#include "G4SDManager.hh"
#include "G4Step.hh"
#include "G4SteppingManager.hh"
#include "G4StepPoint.hh"
#include "G4Track.hh"
#include "G4TrackStatus.hh"
#include "G4VPhysicalVolume.hh"

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

LXeSteppingAction::LXeSteppingAction(LXeEventAction* ea)
  : fOneStepPrimaries(false)
  , fEventAction(ea)
{
  fSteppingMessenger = new LXeSteppingMessenger(this);

  fExpectedNextStatus = Undefined;
}

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

LXeSteppingAction::~LXeSteppingAction() { delete fSteppingMessenger; }

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

void LXeSteppingAction::UserSteppingAction(const G4Step* theStep)
{
  G4Track* theTrack = theStep->GetTrack();

  if(theTrack->GetCurrentStepNumber() == 1)
    fExpectedNextStatus = Undefined;

  LXeUserTrackInformation* trackInformation =
    (LXeUserTrackInformation*) theTrack->GetUserInformation();
```

```
G4StepPoint* thePrePoint    = theStep->GetPreStepPoint();
G4VPhysicalVolume* thePrePV = thePrePoint->GetPhysicalVolume();

G4StepPoint* thePostPoint    = theStep->GetPostStepPoint();
G4VPhysicalVolume* thePostPV = thePostPoint->GetPhysicalVolume();

G4OpBoundaryProcessStatus boundaryStatus           = Undefined;
static G4ThreadLocal G4OpBoundaryProcess* boundary = nullptr;

// find the boundary process only once
if(!boundary)
{
  G4ProcessManager* pm =
    theStep->GetTrack()->GetDefinition()->GetProcessManager();
  G4int nprocesses    = pm->GetProcessListLength();
  G4ProcessVector* pv = pm->GetProcessList();
  for(G4int i = 0; i < nprocesses; ++i)
  {
    if((*pv)[i]->GetProcessName() == "OpBoundary")
    {
      boundary = (G4OpBoundaryProcess*) (*pv)[i];
      break;
    }
  }
}

if(theTrack->GetParentID() == 0)
{
  // This is a primary track

  G4TrackVector* fSecondary = fpSteppingManager->GetfSecondary();
  G4int tN2ndariesTot       = fpSteppingManager->GetfN2ndariesAtRestDoIt() +
                      fpSteppingManager->GetfN2ndariesAlongStepDoIt() +
                      fpSteppingManager->GetfN2ndariesPostStepDoIt();
  if(!fEventAction->IsConvPosSet() && tN2ndariesTot > 0)
  {
    for(size_t lp1 = (*fSecondary).size() - tN2ndariesTot;
        lp1 < (*fSecondary).size(); ++lp1)
    {
      const G4VProcess* creator = (*fSecondary)[lp1]->GetCreatorProcess();
      if(creator)
      {
```

```
          G4String creatorName = creator->GetProcessName();
          if(creatorName == "phot" || creatorName == "compt" ||
             creatorName == "conv")
          {
            fEventAction->SetConvPos((*fSecondary)[lp1]->GetPosition());
          }
        }
      }
    }

    if(fOneStepPrimaries && thePrePV->GetName() == "scintillator")
      theTrack->SetTrackStatus(fStopAndKill);
  }

  if(!thePostPV)
  {  // out of world
    fExpectedNextStatus = Undefined;
    return;
  }

  if(theTrack->GetDefinition() == G4OpticalPhoton::OpticalPhotonDefinition())
  {
    // Optical photon only

    if(thePrePV->GetName() == "Slab")
      trackInformation->SetForceDrawTrajectory(true);
    else if(thePostPV->GetName() == "expHall")
      theTrack->SetTrackStatus(fStopAndKill);

    if(thePostPoint->GetProcessDefinedStep()->GetProcessName() ==
       "OpAbsorption")
    {
      fEventAction->IncAbsorption();
      trackInformation->AddTrackStatusFlag(absorbed);
    }

    boundaryStatus = boundary->GetStatus();

    if(thePostPoint->GetStepStatus() == fGeomBoundary)
    {
      if(fExpectedNextStatus == StepTooSmall)
      {
        if(boundaryStatus != StepTooSmall)
        {
```

```
          G4ExceptionDescription ed;
          ed << "LXeSteppingAction::UserSteppingAction(): "
             << "No reallocation step after reflection!" << G4endl;
          G4Exception("LXeSteppingAction::UserSteppingAction()", "LXeExpl01",
                      FatalException, ed,
                      "Something is wrong with the surface normal or geometry");
        }
      }
      fExpectedNextStatus = Undefined;
      switch(boundaryStatus)
      {
        case Absorption:
          trackInformation->AddTrackStatusFlag(boundaryAbsorbed);
          fEventAction->IncBoundaryAbsorption();
          break;
        case Detection:
        {
          G4SDManager* SDman = G4SDManager::GetSDMpointer();
          G4String sdName    = "/LXeDet/pmtSD";
          LXePMTSD* pmtSD    = (LXePMTSD*) SDman->FindSensitiveDetector(sdName);
          if(pmtSD)
            pmtSD->ProcessHits_boundary(theStep, nullptr);
          trackInformation->AddTrackStatusFlag(hitPMT);
          break;
        }
        case FresnelReflection:
        case TotalInternalReflection:
        case LambertianReflection:
        case LobeReflection:
        case SpikeReflection:
        case BackScattering:
          trackInformation->IncReflections();
          fExpectedNextStatus = StepTooSmall;
          break;
        default:
          break;
      }
      if(thePostPV->GetName() == "sphere")
        trackInformation->AddTrackStatusFlag(hitSphere);
    }
  }
}
```

LXeSteppingMessenger.cc

```
#include "LXeSteppingMessenger.hh"

#include "LXeSteppingAction.hh"

#include "G4UIcmdWithABool.hh"
#include "G4UIdirectory.hh"

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

LXeSteppingMessenger::LXeSteppingMessenger(LXeSteppingAction* step)
  : fStepping(step)
{
  fOneStepPrimariesCmd = new G4UIcmdWithABool("/LXe/oneStepPrimaries", this);
  fOneStepPrimariesCmd->SetGuidance(
    "Only allows primaries to go one step before being killed.");
}

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

LXeSteppingMessenger::~LXeSteppingMessenger() { delete fOneStepPrimariesCmd; }

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

void LXeSteppingMessenger::SetNewValue(G4UIcommand* command, G4String newValue)
{
  if(command == fOneStepPrimariesCmd)
  {
    fStepping->SetOneStepPrimaries(
      fOneStepPrimariesCmd->GetNewBoolValue(newValue));
  }
}
```

LXeTrackingAction.cc

```
#include "LXeTrackingAction.hh"

#include "LXeDetectorConstruction.hh"
#include "LXeTrajectory.hh"
#include "LXeUserTrackInformation.hh"

#include "G4OpticalPhoton.hh"
#include "G4Track.hh"
#include "G4TrackingManager.hh"

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

LXeTrackingAction::LXeTrackingAction() {}

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

void LXeTrackingAction::PreUserTrackingAction(const G4Track* aTrack)
{
  // Let this be up to the user via vis.mac
  //  fpTrackingManager->SetStoreTrajectory(true);

  // Use custom trajectory class
  fpTrackingManager->SetTrajectory(new LXeTrajectory(aTrack));

  // This user track information is only relevant to the photons
  fpTrackingManager->SetUserTrackInformation(new LXeUserTrackInformation);
}

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

void LXeTrackingAction::PostUserTrackingAction(const G4Track* aTrack)
{
  LXeTrajectory* trajectory =
    (LXeTrajectory*) fpTrackingManager->GimmeTrajectory();
  LXeUserTrackInformation* trackInformation =
    (LXeUserTrackInformation*) aTrack->GetUserInformation();

  // Let's choose to draw only the photons that hit the sphere and a pmt
  if(aTrack->GetDefinition() == G4OpticalPhoton::OpticalPhotonDefinition())
  {
```

```
    const G4VProcess* creator = aTrack->GetCreatorProcess();
    if(creator && creator->GetProcessName() == "OpWLS")
    {
      trajectory->WLS();
      trajectory->SetDrawTrajectory(true);
    }

    if(LXeDetectorConstruction::GetSphereOn())
    {
      if((trackInformation->GetTrackStatus() & hitPMT) &&
         (trackInformation->GetTrackStatus() & hitSphere))
      {
        trajectory->SetDrawTrajectory(true);
      }
    }
    else
    {
      if(trackInformation->GetTrackStatus() & hitPMT)
        trajectory->SetDrawTrajectory(true);
    }
  }
  // draw all other (not optical photon) trajectories
  else
    trajectory->SetDrawTrajectory(true);

  if(trackInformation->GetForceDrawTrajectory())
    trajectory->SetDrawTrajectory(true);
}
```

LXeUserTrackInformation.cc

```
#include "LXeUserTrackInformation.hh"

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

LXeUserTrackInformation::LXeUserTrackInformation()
  : fStatus(active)
  , fReflections(0)
  , fForcedraw(false)
{}

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

LXeUserTrackInformation::~LXeUserTrackInformation() {}

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

void LXeUserTrackInformation::AddTrackStatusFlag(int s)
{
  if(s & active)           // track is now active
    fStatus &= ~inactive;  // remove any flags indicating it is inactive
  else if(s & inactive)    // track is now inactive
    fStatus &= ~active;    // remove any flags indicating it is active
  fStatus |= s;            // add new flags
}
```

Bibliografía

[1] D. Michael. Update on MINOS. *Nucl. Phys. B Proc. Suppl.*, 66:432–435, 1998.

[2] A. G. Cocco. The OPERA experiment at Gran Sasso. *Nucl. Phys. B Proc. Suppl.*, 85:125–128, 2000.

[3] S. Agostinelli et al. GEANT4: A simulation toolkit. *Nucl. Instrum. Meth.*, A506:250–303, 2003.

[4] Enrico Fermi. On the Origin of the Cosmic Radiation. *Phys. Rev.*, 75:1169–1174, 1949.

[5] John Linsley. Evidence for a primary cosmic-ray particle with energy 10**20-eV. *Phys. Rev. Lett.*, 10:146–148, 1963.

[6] Pierre Auger, P. Ehrenfest, R. Maze, J. Daudin, and A. Freon Robley. Extensive cosmic ray showers. *Rev. Mod. Phys.*, 11:288–291, 1939.

[7] David T. Wilkinson. Measurement of the Cosmic Microwave Background at 8.56-mm Wavelength. *Phys. Rev. Lett.*, 19:1195–1198, 1967.

[8] Thomas K. Gaisser, Ralph Engel, and Elisa Resconi. *Cosmic Rays and Particle Physics*. 2016.

[9] R.L. Workman et al. Review of Particle Physics. to be published (2022).

[10] J. Hamilton, W. Heitler, and H. W. Peng. THEORY OF COSMIC RAY MESONS. *Phys. Rev.*, 64:78–94, 1943.

[11] J. Matthews. A description of some ultrahigh energy cosmic rays observed with the Pierre Auger Observatory. In *29th International Cosmic Ray Conference*, 7 2005.

[12] P. A. Zyla et al. Review of Particle Physics. *PTEP*, 2020(8):083C01, 2020.

[13] Donald E. Groom, Nikolai V. Mokhov, and Sergei I. Striganov. Muon stopping power and range tables 10-MeV to 100-TeV. *Atom. Data Nucl. Data Tabl.*, 78:183–356, 2001.

[14] Jovana Knezevic, Dusan Mrdja, Kristina Bikit, Jan Hansman, Sofija Forkapic, Danijel Velimirovic, and Istvan Bikit. Time characterization of cosmic-ray induced events in HPGe detector by Monte Carlo simulations. *Nucl. Instrum. Meth. A*, 1032:166624, 2022.

[15] Alan D. Bross et al. Tomographic Muon Imaging of the Great Pyramid of Giza. 2 2022.

[16] D. Carbone, D. Gibert, J. Marteau, M. Diament, L. Zuccarello, and E. Galichet. An experiment of muon radiography at Mt Etna (Italy). *Geophys. J. Int.*, 196:633–643, 2014.

[17] Subhendu Das, Sridhar Tripathy, Sandip Sarkar, Nayana Majumdar, and Supratik Mukhopadhyay. A Novel Readout Scheme for Muon Tomography Application in Material Identification. 5 2022.

[18] N. Lesparre, D. Gibert, J. Marteau, Y. Declais, D. Carbone, and E. Galichet. Geophysical muon imaging: feasibility and limits. *Geophys. J. Int.*, 183:1348–1361, 2010.

[19] M. Ambrosio et al. Low-energy atmospheric muon neutrinos in MACRO. *Phys. Lett. B*, 478:5–13, 2000.

[20] Karl-Heinz Kampert, Alan A. Watson, and Alan A. Watson. Extensive Air Showers and Ultra High-Energy Cosmic Rays: A Historical Review. *Eur. Phys. J. H*, 37:359–412, 2012.

[21] Spencer Nicholas Gaelan Axani. The Physics Behind the CosmicWatch Desktop Muon Detectors. Master's thesis, MIT, 2019.

[22] Javier Tiffenberg et al. DAMIC: a novel dark matter experiment. In *33rd International Cosmic Ray Conference*, page 1243, 10 2013.

[23] W. Bugg, Yu. Efremenko, and S. Vasilyev. Large Plastic Scintillator Panels with WLS Fiber Readout; Optimization of Components. *Nucl. Instrum. Meth. A*, 758:91–96, 2014.

[24] S. Vinogradov. Skewness-based characterization of silicon photomultipliers. *Eur. Phys. J. C*, 82(5):490, 2022.

[25] Photonics. Detectors: Guideposts on the road to selection. `https://www.photonics.com/Articles/Detectors_Guideposts_on_the_Road_to_Selection/a25535`.

[26] Photonics. Detectors: Guideposts on the road to selection. `https://www.hamamatsu.com/us/en/product/optical-sensors/mppc/related_documents.html`.

[27] Markus Ackermann et al. High-Energy and Ultra-High-Energy Neutrinos. 3 2022.

[28] F. A. F. Fraga, S. T. G. Fetal, M. M. F. R. Fraga, E. F. S. Balau, L. M. S. Margato, R. Ferreira-Marques, A. J. P. L. Policarpo, and F. Sauli. The scintillation of GEMS coated with wavelength shifters. *Nucl. Instrum. Meth. A*, 525:57–61, 2004.

[29] William R Leo. *Techniques for nuclear and particle physics experiments: a how-to approach; 2nd ed.* Springer, Berlin, 1994.

[30] P. A. Cherenkov. Visible luminescence of pure liquids under the influence of γ-radiation. *Dokl. Akad. Nauk SSSR*, 2(8):451–454, 1934.

[31] Rayleigh. The Problem of the Random Walk. *Nature*, 72(1866):318–318, 1905.

[32] Yan Benhammou, Erez Etzion, Gilad Mizrachi, Meny Raviv Moshe, Yiftah Silver, and Igor Zolkin. Muon Detector for Underground Tomography. 5 2022.

[33] S. W. Ham, J. O. Im, E. J. Yoo, and S. K. Oh. Higgs bosons of a supersymmetric E(6) model at the Large Hadron Collider. *JHEP*, 12:017, 2008.

[34] Cern. Tutorials. `https://geant4.web.cern.ch/node/597#intro`.

[35] 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 root"[software], release vx.yy/zz, dd/mm/yyyy. `https://root.cern.ch/`.

[36] Ta-Pei Cheng and Ling-Fong Li. *Gauge Theory of Elementary Particles*. Oxford University Press, 2000.

[37] Chris Quigg. *Gauge Theories of the Strong, Weak, and Eletromagnetic Interactions*. Princeton University Press, 2013.

[38] David Griffiths. *Introduction to Elementary Particles*. Wiley-VHC, 2008.

[39] C. S. Wu, E. Ambler, R. W. Hayward, D. D. Hoppes, and R. P. Hudson. Experimental Test of Parity Conservation in β Decay. *Phys. Rev.*, 105:1413–1414, 1957.

[40] R. Slansky. Group Theory for Unified Model Building. *Phys. Rept.*, 79:1–128, 1981.

[41] N. M. Budnev et al. The tunka experiment: towards a 1-km^2 eas Cerenkov light array in the tunka valley. In *29th International Cosmic Ray Conference*, 11 2005.

[42] Andy Beretvas. Visualization drivers for Geant4. 10 2005.