THESIS

DESIGN OF A COMPTON SCATTER BASED RADIATION TRACKING SYSTEM

Submitted by

Heather Healy

Department of Environmental and Radiological Health Sciences

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Spring 2015

ABSTRACT

DESIGN OF A COMPTON SCATTER BASED RADIATION TRACKING SYSTEM

Gamma spectroscopy is one of the most common techniques used for the detection of radiologic materials. This technology is deployed in a variety of scenarios such as emergency response, monitoring, and the recovery of lost, stolen, or otherwise unaccounted radiologic material. In most practical scenarios, it is useful to know the location of a source in relation to a detector, in addition to the classic output from gamma spectrometers such as decay rate and energy peak information. In collaboration with the Remote Sensing Laboratory (RSL) at Andrew's Air Force Base, a novel detector design by RSL, which utilizes a 360^o detectable range in order to increase the probability of remote detection, was investigated for the possibility to recreate source location information from Compton scattering events within the detector. A recreation of this novel detector is simulated using Geant4 to determine the optimal dimensions of sodium iodide detectors that produce the most single Compton scattering events in order to facilitate source location through the back-projection of Compton scattering angles. The optimal detector dimensions are determined by maximizing the number of single Compton scatter events and minimizing the percentage of Compton events that undergo multiple successive scatters in detectors of varying thicknesses and lengths. The optimal detector thickness was chosen to be 1.88 in, and the optimal detector length was chosen to be 4 to 4.5 in. In future projects, these optimized detectors can be used to apply suggested back-projection algorithms in order to determine the feasibility and functionality of this detector design for the purpose of radiologic source location.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

Motivation

Current radiologic detection methods rely heavily on the detection of gamma photons to identify and analyze radiological sources that are lost, stolen, or otherwise unaccounted for by the owner. The Remote Sensing Laboratory (RSL) at Andrews Air Force Base is one facility that specializes in the detection of radiologic material and continuously contributes to the evolution of radiation detection techniques. In a recent project, a novel detector unit was designed for 360^o remote gamma spectroscopy and detection (Kiser, 2010). A follow up investigation of the detector unit’s ability to locate radiologic sources, in addition to its primary functions, was requested.

Ionizing Particles

Gamma photons are one of the easiest types of radiation to detect from a distance. Alpha particles (commonly emitted from heavy nuclei) deposit their energy within very short distances from where they are emitted (on the order of mm). A piece of paper is enough to shield alpha particles. Beta particles (electrons or positrons) travel slightly farther than alpha particles (on the order of cm), and are emitted from a wide variety of radionuclides, but they can be shielded with a stack of paper making both alpha and beta particles difficult to detect from a distance. High energy gamma and X-ray photons are capable of traveling longer distances than alpha and beta particles (on the order of m). High energy photons require more robust shielding materials such as lead to reduce the transmission of these particles, making the detection of gamma and X-ray photons more likely at distances even if they are shielded.

It is a common misconception that X-ray and gamma photons are different types of radiation with different energy levels. X-rays and gammas are both photons and can both be produced along the same spectrum of energies. The difference between the two designations for these photons comes from how they are produced. Gamma photons are emitted from the nucleus of an atom during radioactive decay or after nuclear excitation events. Gamma photon energies are discrete and specific to the radionuclide from which they are emitted (Cember & Johnson, 2009). Conversely, there are two processes that produce X-rays. The first is when an electron transitions from one energy state to a lower energy state within an atom's electron cloud. These are characteristic X-rays because the amount of energy released by the transitioning of the electron is characteristic to the difference in energy between the two energy states of the atom and by that characteristic to the element associated with that atom (Cember & Johnson, 2009). The second type of X-ray is the bremsstrahlung X-ray, which is German for breaking radiation (Cember & Johnson, 2009). Bremsstrahlung X-rays are produced when charged particles traveling at high velocities suddenly experience a change in velocity, such as by changing direction. This can occur if an electron is traveling at a high velocity near an atom. The electron is drawn toward the atoms' nucleus because of their electrostatic attraction due to the difference in charge (nuclei are positive and electrons are negative). This attraction causes the electron to very suddenly change its trajectory and continue traveling past the nucleus along a new trajectory (like a comet and the sun). The sudden change in velocity of the electron from the direction change results in the emission of energy in the form of a bremsstrahlung X-ray (Cember & Johnson, 2009). Characteristic and bremsstrahlung X-rays are emitted from outside the nucleus in the electron cloud, not from the nucleus like gamma photons. Both X-ray and gamma photons can be used in the detection of radiological sources.

Inorganic Scintillators

The novel detector designed by the RSL is a sodium iodide detector, which is a type of inorganic scintillator. Inorganic scintillators are composed of two major components: a scintillator, which is a crystalline structure (such as NaI), and a photomultiplier tube (PM tube). Photons enter the detector and excite electrons within the crystalline lattice of the scintillator. An excitation event occurs when a particle imparts energy onto an electron that is less than the total binding energy, so the electron remains within the crystal lattice. The crystalline lattice has different energy states to which electrons within the lattice can be excited (Knoll, 2010). Typically, these energy states are referred to as the excited states (higher energy) and the ground state. The ground state is the lowest possible energy state (Cember & Johnson, 2009). Electrons within the crystalline lattice will always transition to the lowest possible energy state because at that state the force acting on the electron is zero making this the most stable state.

Different atoms and molecular structures can have a variety of different energy states. The difference in the energy states determines what process an atom can undergo to dissipate the excess energy and return to the ground state where the electronic configuration is most stable. Pure sodium iodide crystals have very wide energy gaps (the difference between energy states) meaning that the electrons must dissipate a large quantum of energy in order to transition back to the ground state. In a sodium iodide crystal this occurs by the emission of an X-ray photon (high energy) (Knoll, 2010). Scintillator detectors rely on the production of visible light to propagate the occurrence of an ionization event within the detector. Therefore, the production of X-rays (not visible) by the excited scintillator is not efficient for further propagation of the signal. To combat this issue, impurities are introduced into the crystalline structure. Thallium is commonly used in sodium iodide detectors. Thallium contains different energy states than sodium iodide, so the addition of Thallium adds more possible excitation states to the compound to which electrons can transition within the crystalline lattice. The excitation states of Thallium are lower than those of sodium iodide meaning that less energy needs to be dissipated to return to the ground state. When an electron is excited to one of the thallium excitation states, a visible light photon (lower energy than X-ray) is emitted upon transitioning back to the ground state allowing the propagation of a signal within the detector (Knoll, 2010).

Once light is produced in the scintillator, it is converted to an electronic signal in the PM tube. The PM tube has two main components: a photocathode and an electron multiplier structure. The photocathode converts the visible light photon into low-energy electrons, which are emitted from the photocathode surface. The electrons then drift toward the electron multiplication region, which contains a series of dynodes. When the low-energy electrons strike a dynode, additional electrons are released toward the next dynode. After the electrons have cascaded down the entire series of dynodes, there are approximately 10^7 to 10^{10} electrons. These electrons are funneled to an anode at the end of the PM tube creating a sharp voltage drop. This voltage drop is easily recognized by detector readout equipment (Knoll, 2010).

The benefits of sodium iodide detectors are that they are portable (easy to use in the field), they are relatively low maintenance, they have a high light output and quick response time, and a wide variety of crystal sizes is available (ability to create different size detectors) (Knoll, 2010). Although sodium iodide detectors have worse resolution than other gamma spectrometers, their portability and high light output make them ideal for mobile operations such as those conducted by the Remote Sensing Laboratory.

Location through Compton Scattering

The researchers at RSL intend to not only be able to detect radiologic materials using this detector unit, but also precisely locate the source of radioactivity. Currently, the best method to determine the relative location of a radiologic source with respect to a sodium iodide detector is to define the direction of the strongest signals as the location of the source. Typically, this type of determination will result in an 180^{o} window of possible source direction. Since information regarding distance to the source is generally unknown in detection scenarios, this technique does little to precisely and accurately identify a source location.

A proposed method to precisely identify the location of a radiologic source is through the analysis of Compton scattering within a detector unit (Kiser, 2010). Compton scattering was discovered by Arthur Compton in 1918, while he was a professor at Washington University studying the scattering of X-rays. He observed that X-rays that were scattered after interaction with electrons in a carbon target had longer wavelengths (different energies) than those incident on the target. Further exploration led to the effect being named after him in 1922 (“Arthur”, 2015).

Compton scattering is the elastic collision between a photon and a free electron (or an electron with a very low binding) at rest, which results in the photon and the electron scattering in different directions at new energies. Kinetic energy and momentum are conserved in elastic collisions. Therefore, the initial kinetic energy of the photon and the final kinetic energy of the photon after the collision are related as a function of the angle through which the photon was scattered (Cember & Johnson, 2009).

The relationship between the energies of incident and final photon can be described using Equation 1

$$E' = \frac{E}{1+\left(\frac{E}{m_0c^2}\right)(1-\cos\theta)}, \qquad (1)$$

where E’ is the energy of the photon after collision, E is the energy of the incident photon, and m_0c^2 is the rest mass of the electron, 511 keV (Cember & Johnson, 2009).

This methodology could be used to determine the location of a radiologic source if the energies of an incident and the scattered photon are known. It is possible to obtain this information from a multi-detector system. A multi-detector system is a detector unit composed of any number of individual sodium iodide detectors, each with their own photomultiplier tube. Ideally, a detector unit covers a 360^o range for the greatest detection efficiency as modeled by the RSL detector unit design (Kiser, 2010).

With this system in place, an incoming photon could collide with an electron in the crystalline structure in one detector and undergo a Compton scatter into a second detector within the unit. The energy of the scattered photon would be equal to the amount of energy deposited in the second detector, while the incident photon energy would be equal to the sum of the energies deposited in the first and second detectors due to the conservation of kinetic energy in elastic collisions (Cember & Johnson, 2009). This information could be used to calculate the scattering angle of the incident photon.

Coincidence Intervals

In order to determine which excitation events are related via Compton scattering within the detector unit, coincidence intervals can be used (Kiser, 2010). A coincidence interval is a time window set on detectors to identify ionization events that occur within a very short amount of time of each other, typically on the order of microseconds or nanoseconds. If two events occur within the coincidence interval they are said to be related by an interaction (in this case Compton scattering) and not due to the chance that two individual photons interacted at or near the same time. The time window used for coincidence intervals is determined based on the detector set up and equipment. Most manufactures offer guidance on the appropriate coincidence interval time window depending on the detector and system in use (“Timing and Coincidence”, n.d.).

Photon Interaction with Matter

Photons interact differently depending on their energy and what type of material they pass through. In air (assuming sea level and 15^{o}C), a 100 keV photon travels an average of 53 m before an interaction with the air takes place, while a 1 MeV photon will travel an average of 130 m (Johnson, 2012). The average distance a photon travels in a material before an interaction takes place is called the mean free path. The mean free path is important because it quantifies a particles probability to transmit through materials without experiences a scattering event. The mean free path is obtained from the linear attenuation coefficient, which is defined as the “probability per unit path length that an ionizing particle interacts” (Knoll, 2010). The mean free path is equal to the inverse of the linear attenuation coefficient. The linear attenuation coefficient for a material is determined by observing how many photons enter a material of a certain thickness and how many of those photons pass through the material without interaction as shown in Equation 2.

$$\frac{N}{N_0} = e^{-\mu x} \, , \tag{2}$$

where N is the number of un-scattered photons exiting an absorber material of thickness x, N_0 is the number of photons incident on the absorber, and μ is the linear attenuation coefficient (Cember & Johnson, 2009). The quantity $e^{-\mu x}$ quantifies the fraction of photons that should be expected to transmit through the absorber without interaction (Cember & Johnson, 2009). If μx = 1 then the expected percentage of un-scattered photons traveling through the absorber material is 37%. At μx = 2 the percentage of un-scattered photons is 14%.

Poisson Distribution

The Poisson distribution was originally developed as an approximation of the binomial distribution to describe larger sample sets of random processes (Turner & Downing, 2012). Today, the Poisson distribution is used to describe "all random processes that occur with a probability that is both small and constant" (Turner & Downing, 2012). This applies to processes that meet the following criteria (1) events are independent of one another, (2) the outcomes are whole numbers (success or no success), (3) the frequency of a success is very small in relation to the total sample size, (4) and the probability of two successes within a short time interval is negligible (Turner & Downing, 2012).

Radioactive decay is one of the random processes best described by the Poisson distribution (Turner & Downing, 2012). Radioactive decay is a completely random process. The decay of one atom has no effect on the decay of any other atom. The outcome of the radioactive decay process is either a decay or no decay in a given time interval. The frequency of radioactive decay is very small relative to the number of atoms that could decay, and the probability of two atoms decaying simultaneously is extremely small.

Similarly, photons that undergo Compton scattering are also well described within the confines of the Poisson distribution. Photons can either Compton scatter or not Compton scatter in material. If one photon undergoes a Compton scatter, it has no impact on the probability that any other photon will Compton scatter. The expected number of Compton scattering events is very small in relation to the total number of photons that interact with matter, and the probability of two Compton scattering events occurring at the same time is extremely small (Nelson & Reilly, n.d.). For these reasons, the Poisson distribution can be used to analyze Compton scattering events in sodium iodide detectors.

When analyzing data containing a small number of successes, it is important to quantify the uncertainty associated with the analysis results. This uncertainty often is expressed in terms of the standard deviation. The standard deviation is an estimator for the true sample variance. The standard deviation, σ, using Poisson statistics is

$$\sigma_A = \sqrt{A}, \tag{3}$$

where, A is the number of successes (Turner & Downing, 2012). When multiple independent measurements are taken, each measurement has its own standard deviation. In some scenarios, it is necessary to combine several independent quantities or variables to obtain a dependent variable. In this case, the standard deviations from all the multiple measurements are propagated through the use of Equation 4 to estimate the associated uncertainty of the dependent variable.

$${\sigma_T}^2 = \sum_{i=1}^{N} \left(\frac{\partial T}{\partial x_i}\right)^2 \sigma_i^2 \tag{4}$$

where, N is the number of independent variables, T is the quantity of interest, x_i is the ith independent variable (measurement) in T, and σ_i is the associated uncertainty for each x_i value (Turner & Downing, 2012). The proper calculation of the uncertainty related to the measurements of Compton scattering events within the detectors allows for the determination of the precision of the results.

Hypothesis

It is expected that sodium iodide detectors will have an optimal thickness and length for maximizing the occurrence of Compton scattering events that can be used to calculate the location of a radiological source because of the probability of photon interactions in different volumes of sodium iodide absorber material.

This study was designed to determine the optimal thickness and width dimensions for single Compton scattering events in six identical sodium iodide detectors arranged in a hexagonal detection unit. Once the optimization of the thickness and length are determined for the detectors, this system could be used to apply tracking and location algorithms in order to locate radiologic sources through the Compton scattering angles associated with events within the detector unit.

MATERIALS AND METHODS

The Remote Sensing Laboratory provided computer files containing the originally designed hexagonal sodium iodide detector unit and operational environment to be run on Geant4, an open source simulation software (Kiser, 2010). As the provided files were not compatible with the Geant4 system available for this study, a basic model of six sodium iodide detectors arranged in a hexagon was developed to represent a basic model of the original detector design. The Geant4 files for this recreation can be found in Appendix A. A particle gun is modeled in the software to emit 662 keV gammas in order to simulate the presence of a ^{137}Cs source. It is important that only one energy photon be used in this preliminary study to eliminate the presence of multiple, possibly competing variables. ^{137}Cs is ideal for this model because it emits single-energy photons when it undergoes radioactive decay and because it is commonly used in industry. Some examples of ^{137}Cs sources used in industry are moisture-density gauges (construction), leveling gauges to detect liquid flow in pipes and tanks, thickness gauges for materials, well-logging devices in the drilling industry, and medical therapy sources. It is not uncommon for high activity ^{137}Cs sources to become misplaced on construction or drilling sites because of the small size of the source. When this happens, detectors are used to locate the missing source.

The particle gun in the model is placed 40 cm from the detector unit. The particle gun is centered along the vertical axis of the detector unit, and the height of the detectors is kept constant throughout the entire experiment at 16 in, which is based on the original RSL detector design (Kiser, 2010).

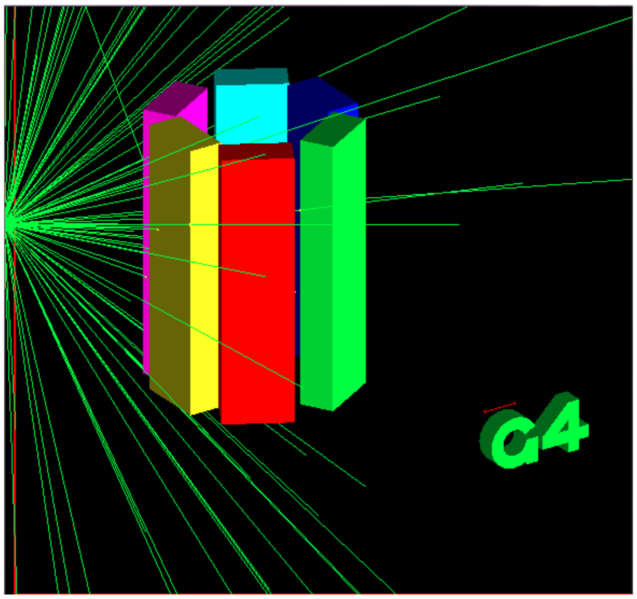

Figure 1 NaI Detectors with Isotropic Hemisphere Source Emitting 662 keV Photons in Geant4

During the first investigation, the length of each detector is held constant at 3 in, which is a common size for sodium iodide crystals. The thickness is set to 1 in in order to determine the linear attenuation coefficient of the modeled sodium iodide crystals.

After the linear attenuation coefficient is determined for the detector simulations and compared to known values for simulation validation, the optimal thickness of the detectors is determined. The thicknesses in Table 1 are tested with relation to the exponent µx from Equation 2.

Table 1 Model Thicknesses of Sodium Iodide Detectors

µx	0.5	0.75	1	1.25	1.5	1.75	2
Thickness (in)	0.75	1.13	1.5	1.88	2.25	2.63	3
Percentage of photons scattered in material	39	53	63	71	78	83	86

At each thickness, 5000 photons are tracked from the particle gun to the detector in order to achieve an uncertainty of 2-3% for the percentage of multiple Compton scatter events as shown in Table 2. As a photon passes through the detector unit, one of three outcomes is recorded: no Compton scatter, single Compton scatter, or multiple Compton scatter. No Compton scatter is defined as a photon that either passes through the detector without interaction, is attenuated completely within one detector, or scatters outside the detector after the first interaction. A single Compton scatter is defined as any photon that has a single hit in one detector resulting in the Compton scatter of the photon into a second detector where the remainder of the photon's energy is deposited. A multiple Compton scatter is defined as a photon that either undergoes more than one Compton scattering event resulting in hits in three or more detectors, the photon scatters outside the detector after deposition of some energy in the second detector following a Compton scattering event, or the photon undergoes two Compton scattering events in the first detector before depositing the rest of its energy in a second detector. Examples of single and multiple Compton scattering events are shown in Figures 2 and 3 respectively.

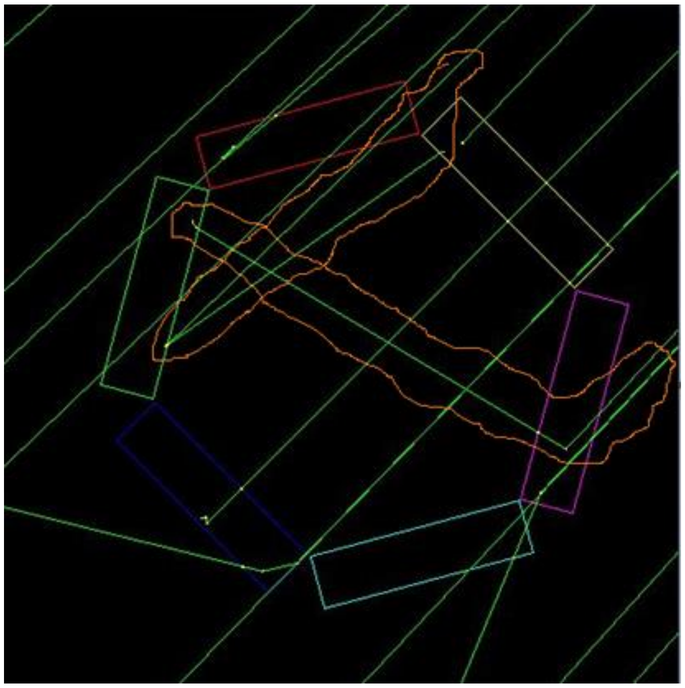

Figure 2 Example of Two Single Compton Scattering Events (Circled in Orange)

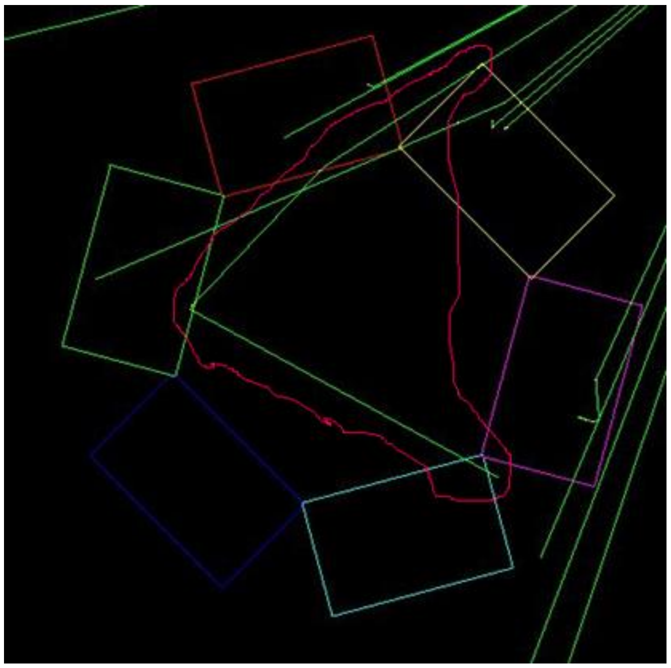

Figure 3 Example of a Multiple Compton Scattering Event (Circled in Red)

Once the optimal thickness is identified based on the highest frequency of single Compton scattering events and the lowest percentage of Compton events that are multiple scatters, the second parameter to be determined is the optimal length of the detectors. The detector height is maintained at 16 in, and the previously determined optimal thickness is maintained constant for all length trials. The modeled detector lengths are 3.0 in, 3.5 in, 4.0 in, 4.5 in, 5.0 in, 7 in, and 10 in. These are commonly manufactured crystal lengths. Seven thousand photons are tracked from the particle gun to the detector for each length in order to achieve an uncertainty of 2-3% for the percentage of multiple Compton scatter events as shown in Table 3. The optimal length is determined using the same criteria as the thickness optimization.

RESULTS

Linear Attenuation Coefficient

The linear attenuation coefficient is measured by tracking 2002 photons from the particle gun to the detector. One thousand and twenty-nine of those particles pass through the detector unit un-scattered. Using Equation 2, the linear attenuation coefficient is 0.6656 in^{-1}. Therefore, in one inch of sodium iodide crystal 51% of the photons that enter the material should pass through without interaction (49% should interact).

Detector Thickness

For each of the thickness trials, approximately 5000 photons are tracked from the particle gun to the detector (some thicknesses had slightly more than 5000 photons tracked to the detector). Sample sizes for each of the thicknesses are normalized by scaling, the number of single and multiple Compton scatter events to 5000 total photons. The results from the thickness trials can be found in Table 2 and Figures 4 and 5. The intervals surrounding the data points in Figures 4 and 5 display the uncertainty associated with the each measurement. These are calculated using Equations 4 and 5.

Table 2 The Number of Single and Multiple Compton Scattering Events at Different Detector Thicknesses Scaled to 5000 Photons

Detector Thickness (µx)	Single Compton Scattering	Multiple Compton Scattering	Percentage of Single Compton Scattering Events Out of 5000 Total Runs	Percentage of Multiple Compton Scattering Events Out of 5000 Total Runs	Percentage of Multiple Compton Scatterings Out of all Compton	Relative Uncertainity of Percentage (%)
0.5	367	268	7.3	5.36	42.2	2.0
0.75	350	223	7	4.5	38.9	2.0
1	280	159	5.6	3.2	36.2	2.3
1.25	249	120	5	2.4	32.5	2.4
1.5	219	105	4.4	2.1	32.4	2.6
1.75	175	91	3.5	1.8	34.2	2.9
2	156	76	3.1	1.5	32.8	3.1

The percentage of Compton events that are multiple scatters, f, is calculated using Equation 5 where M is the number of multiple Compton scattering events and S is the number of single Compton scattering events.

$$\mathrm{f} = \frac{M}{S+M} \tag{5}$$

S and M are independent of each other and have independent standard deviations. Error propagation is used to assess the relative uncertainty of the quantity calculated in Equation 5. The standard deviation of a single independent variable for a quantity described by Poisson statistics is shown in Equation 3. This can be applied by using the error propagation in Equation 4 where

$$\frac{\delta f}{dM} = \frac{S}{(S+M)^2} \quad and \quad \frac{\delta f}{dS} = -\frac{M}{(S+M)^2} \ ,$$

$$\sigma_f^2 = \left(\frac{S}{(S+M)^2}\right)^2 \sqrt{M}^2 + \left(-\frac{M}{(S+M)^2}\right)^2 \sqrt{S}^2 \ ,$$

$$\sigma_f^2 = \frac{S^2 M}{(S+M)^4} + \frac{M^2 S}{(S+M)^4} \, ,$$

$$\sigma_f^2 = \frac{MS}{(S+M)^3} \, .$$

This method is used to calculate the uncertainty for the percentage of Compton events that are multiple scatters in the detectors during the thickness and length trials.

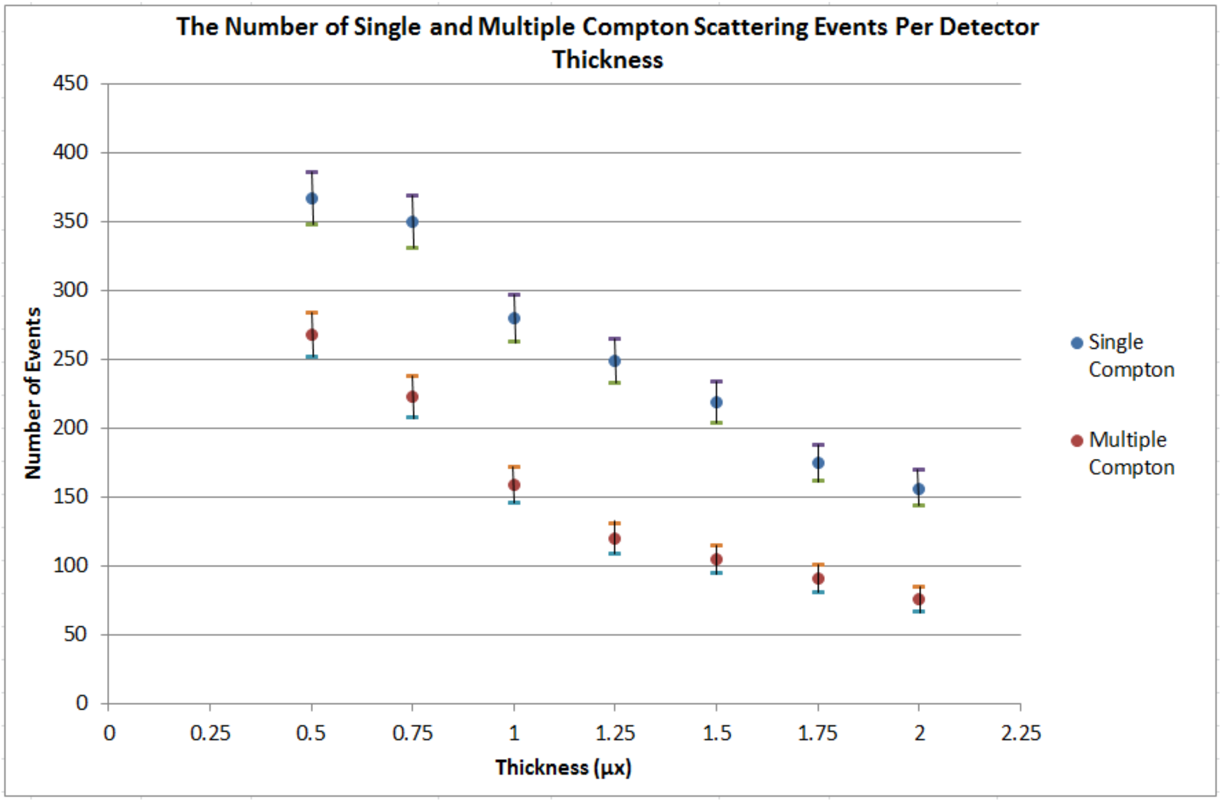

Figure 4 The Number of Single and Multiple Compton Scattering Events per Detector Thickness Scaled to 5000 Photons

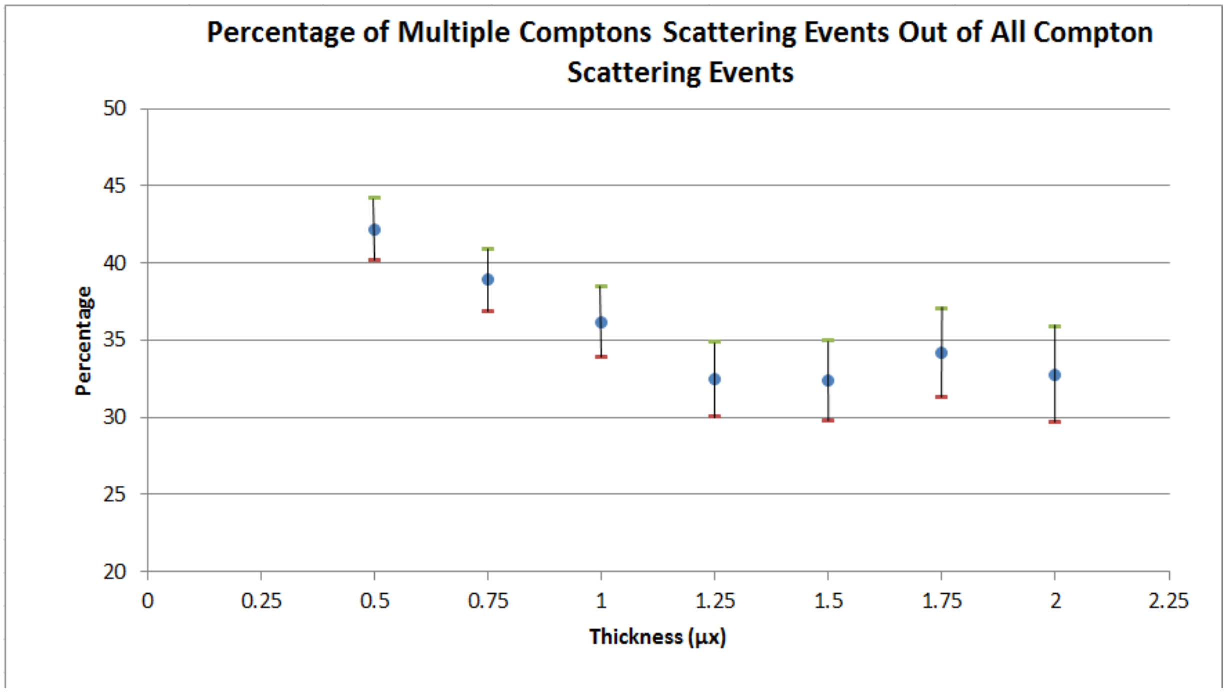

Figure 5 The Percentage of Multiple Compton Scattering Events per Detector Thickness Scaled to 5000 Photons

The 1.25 μx (1.88 in) thickness has the highest number of single Compton scatter events that corresponds with the lowest percentage of multiple Compton events and has the narrowest relative uncertainty.

Detector Length

The optimization for the detector length is conducted with the 1.25 μx (1.88 in) thickness. For each of the length trials, 7000 photons are tracked from the particle gun to the detector. The results of the detector length trials are shown in Table 3 and Figures 6 and 7. The intervals surrounding the data points in Figures 6 and 7 display the uncertainty associated with the each measurement. These are calculated using Equations 3 and 4.

Table 3 The Number of Single and Multiple Compton Scattering Events at Different Detector Lengths Scaled to 7000 Photons

Detector Length (in)	Single Compton Scattering	Multiple Compton Scattering	Percentage of Single Compton Scattering Events Out of 7000 Total Runs	Percentage of Multiple Compton Scattering Events Out of 5000 Total Runs	Percentage of Multiple Compton Scatterings Out of all Compton	Relative Uncertainity of Percentage (%)
3	350	162	5.0	2.3	31.6	2.1
3.5	354	164	5.1	2.3	31.7	2.0
4	315	116	4.5	1.7	26.9	2.1
4.5	301	128	4.3	1.8	29.8	2.2
5	271	104	3.9	1.5	27.7	2.3
7	231	88	3.3	1.3	27.6	2.5
10	174	55	2.5	0.08	24.0	2.8

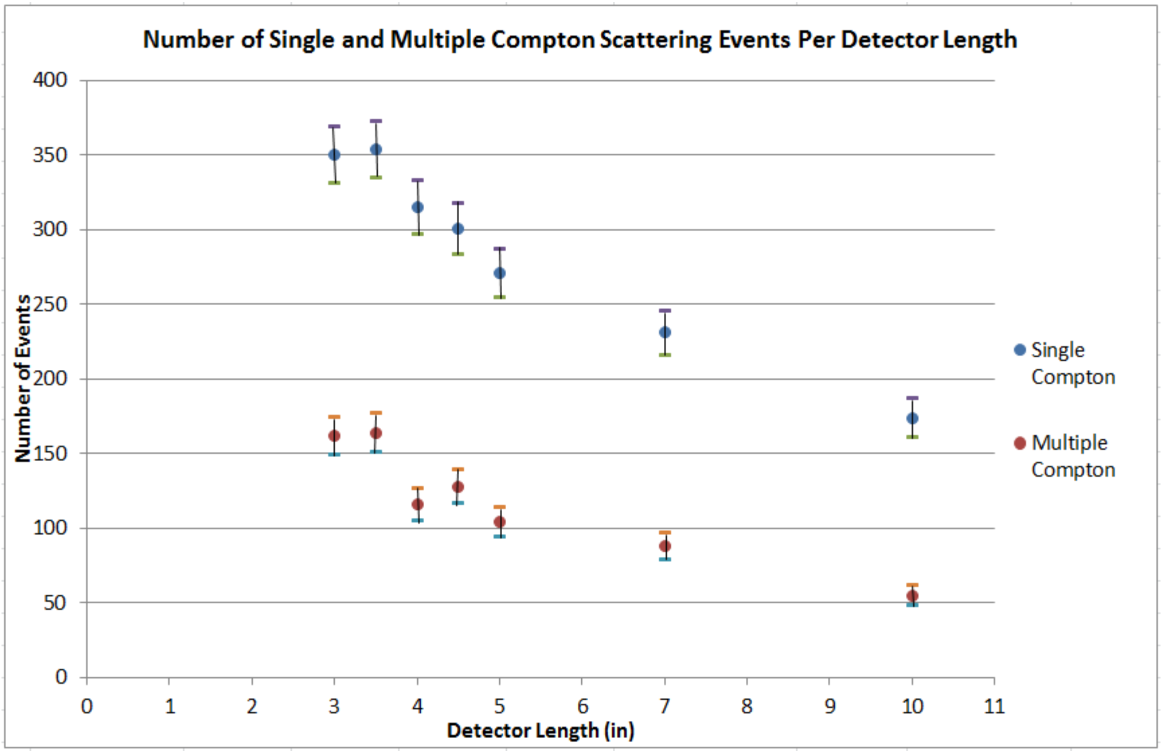

Figure 6 The Number of Single and Multiple Compton Scattering Events per Detector Length Scaled to 7000 Photons

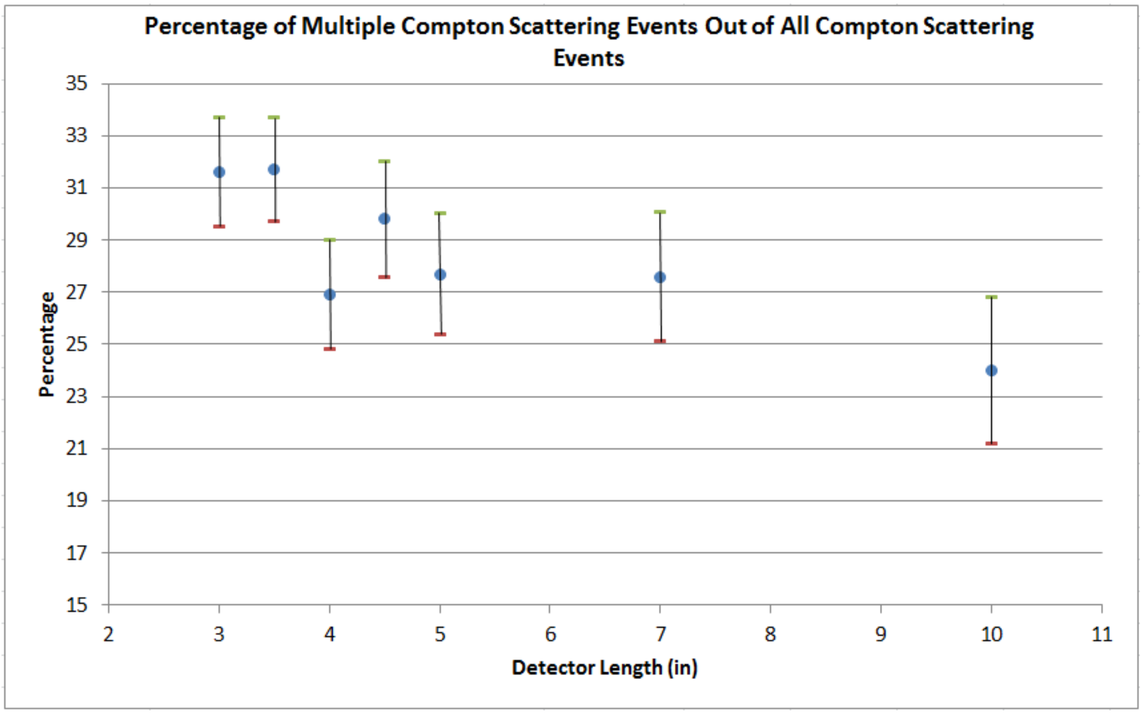

Figure 7 The Percentage of Multiple Compton Scattering Events per Detector Length Scaled to 7000 Photons

Looking at Figure 7, all of the confidence intervals for lengths greater than four inches overlap, indicating that there is no real difference in the percentage of multiple Compton events between these lengths. Based on that information, the optimal detector length is chosen to be 4 in for this study because the 4 in detector has the highest number of single Compton scattering events for detector lengths of 4 in and greater.

DISCUSSION

Linear Attenuation Coefficient

The linear attenuation coefficient is calculated to validate the properties of the sodium iodide absorber material in the simulation against known experimental values. The calculated linear attenuation coefficient of 0.67 in^{-1} (0.26 cm^{-1}) is compared to the experimental value found in *Nelson and Reilly (n.d.),* which is approximately 0.30 cm^{-1}. This demonstrates that the simulated sodium iodide detectors behave similarly to live experimental results; therefore, the model used in this study appears to be appropriate.

Optimization Criteria and Minimizing Uncertainty

In this experiment, the optimized thickness and length of the detector are defined as the value that has the highest frequency of single Compton scatter events and the lowest percentage of multiple Compton events. These criteria are based on the assumption that single Compton scattering events will produce a more precise angular calculation than the multiple Compton scattering events. As previously stated, a multiple Compton scattering event is defined as either (1) a photon that undergoes more than one Compton scattering event resulting in hits in three or more detectors, (2) a photon scatters outside the detector after deposition of some energy in the second detector following a Compton scattering event, or (3) a photon undergoes two Compton scattering events in the first detector before depositing the rest of its energy in a second detector.

In the first of the multiple Compton scattering scenarios, it is assumed that the coincidence interval is narrow enough only to record the scattering of a photon between two detectors. Under this assumption, if not all of the final photon energy is deposited in the second detector, it is not possible to calculate the scatter angle if the original photon energy is unknown because not all of the photon energy is deposited in the first two detectors. Furthermore, even if the coincidence interval is long enough to associate the scattering of a photon between three or more detectors, the associated uncertainty with the calculation of the scatter angle would be significantly larger for two or more angles than just one because the uncertainties compound.

One of the sources of uncertainty in the scattering angle calculations comes from the calibration and resolution of the detector (Parra, 2002). The resolution of a detector is a measure of how precisely the detector system reports the actual deposition of energy in the output. Since the Compton scattering equation is dependent on the energies of the incident and final photon, any inaccuracy in the energy measurements will result in an angle calculation with higher uncertainty. Scintillators have poor resolution compared to other types of detectors, and sodium iodide detectors have a higher resolution percentage (lower resolution) than other types of inorganic scintillators, as shown in Figure 8 where the best possible resolution is displayed by the theoretical limit (Knoll, 2010).

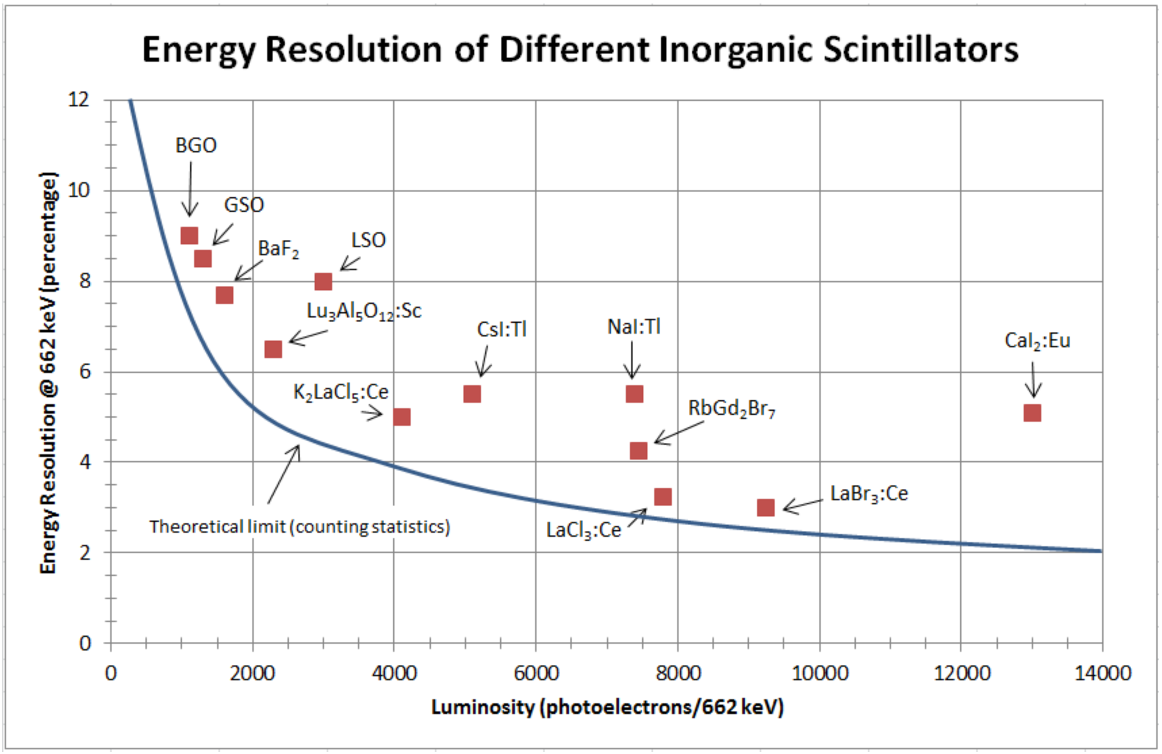

Figure 8 The Energy Resolution at 662 keV of Various Inorganic Scintillators as a Function of Luminosity (Recreated from an Knoll, 2010)

For this reason, only the single Compton scattering events are desirable for the calculation of the scattering angle because the calculation of one angle will decrease the uncertainty of a directional back-projection of the angle from the detector to the possible source location. To minimize the effects of poor resolution of a sodium iodide detector, users should ensure that the detectors are calibrated carefully.

A second source of uncertainty associated with the Compton scattering angular calculation comes from the uncertainty associated with where the photon actually deposits energy within the detector (Parra, 2002). This information is essential for the precise application of the calculated scattering angle. Since current detectors provide poor event location information, it is necessary to minimize the other uncertainties associated with resolution and calibration.

Thickness Optimization

Figure 4 establishes that the number of single and multiple Compton scattering events decreases as the thickness of the detector increases. Also, as the detector thickness increases, it is observed that the number of photons that are completely attenuated or undergo multiple interactions in the first detector increases. In the thinner detectors, it is observed that the majority of the multiple Compton scattering events occur because the photons scatter into three or more detectors. A likely explanation as to why more photons are attenuated or undergo multiple interactions in the thicker detectors, and why photons are more likely to scatter between several of the thinner detectors is because the farther a photon travels in the absorber material, the more probable it is that the photon will interact in the material more than once.

Through examination of Figure 5, it is apparent that the uncertainty intervals associated with the detector thicknesses of 1.25 μx and greater overlap significantly and as a group have a smaller average percentage of multiple Compton events than the detector thicknesses less than 1.25 μx. This indicates that there is not enough evidence to suggest that there is a true difference in the percentage of multiple Compton events for detector thicknesses larger than and equal to 1.25 μx, but the detector thicknesses greater than and equal to 1.25 μx have a lower percentage of multiple Compton scattering events than the detector thicknesses of 0.5-1 μx. The 1.25 μx detector is chosen as the optimal thickness because it has the highest number of single Compton scattering events when compared to the detector thicknesses larger than 1.25 μx.

Length Optimization

As seen in the thickness trials, the total number of single and multiple Compton scattering events decreases as the length of the detectors increases. This is likely because as the length of the detectors increases, the diameter of the detector unit becomes larger, which increases the probability that a photon will interact in the air before interacting with a second or multiple detector in the unit. This is likely the cause for the decrease in the percentage of multiple Compton scattering events as the length of the detector increases. It also appears that as the detector lengths increase, most photons are completely attenuated by a single detector because of the increased volume of the detector. In the shorter length detectors, photons that scatter along the length of the detector are more likely to leave the detector because there is less absorber material between the scattering point and the edge of the detector. In the longer detector lengths, there is more absorber material between the scattering point and the edge of the detector. The concept is similar to increasing the thickness of the detector. Larger volumes of absorber material increase the attenuation of photons because larger volumes increase the probability of photon interaction.

The uncertainty intervals in Figure 7 for detector lengths 4 in and longer all overlap. This indicates that there is no difference in the true percentage of multiple Compton events for detectors 4 in in length and longer. Since the percentage of multiple Compton events in detectors greater than and equal to 4 in is the same, the determination for the optimal detector length is based on which detector greater than or equal to 4 in has the most single Compton scattering events. As seen in Table 3, the detector lengths of 4 and 4.5 in have the greatest number of single Compton scattering events. Figure 6 shows that the uncertainty intervals associated with the 4 and 4.5 in detector lengths overlap significantly demonstrating that there is no significant difference in these values and both are optimal detector lengths.

Application of the Compton Scatter Angle Calculation

The optimization criteria are selected in order to minimize the uncertainty associated with calculating the Compton scattering angle as discussed above. The Compton scattering angle calculated using Equation 1 is a two dimensional angle associated with the incident and scattered photon in a single plane spanned by the incident and scattered photon trajectory vectors. In order to locate a radiologic source using the Compton scattering method, the calculated scattering angle must be applied to a three dimensional system to be used in real detection scenarios. In a three dimensional system, the Compton scattering angle translates into a cone that extends from the first detector of interaction toward the source at the calculated scattering angle. The half angle, ϴ, of the cone is defined by the uncertainty associated with the scatter angle calculation as seen in Figure 9. The true scattering angle of the photon is contained within the cone.

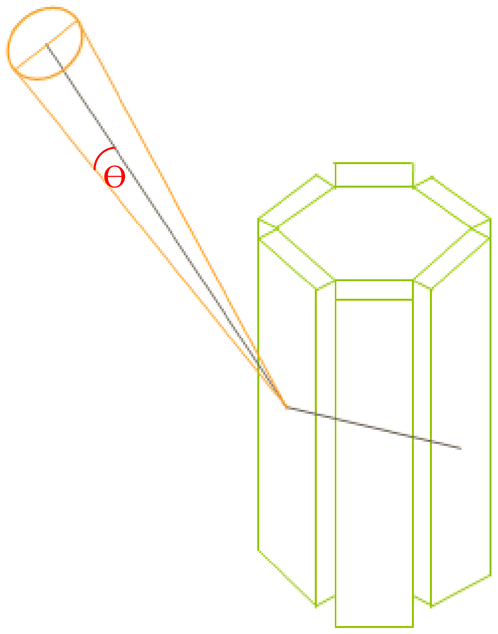

Figure 9 Illustration of the Compton Scattering Angle Associated with a Cone of Angular Uncertainty

The composition of the Compton scattering angle cone does not take into account the incoming direction of the incident photon. For example, if a photon scatters at an angle of 45^{o}, the incident photon could have originated from above, below, to either side of the detector, or anywhere in between before scattering at the calculated angle. The summation of all possible incident photon directions creates a larger cone composed of an infinite number of the same calculated Compton scattering angle with associated uncertainty that rotates around a 360^{o} axis as shown in Figure 10. This is sometimes referred to as the “Compton cone” (Suzuki et al., 2013).

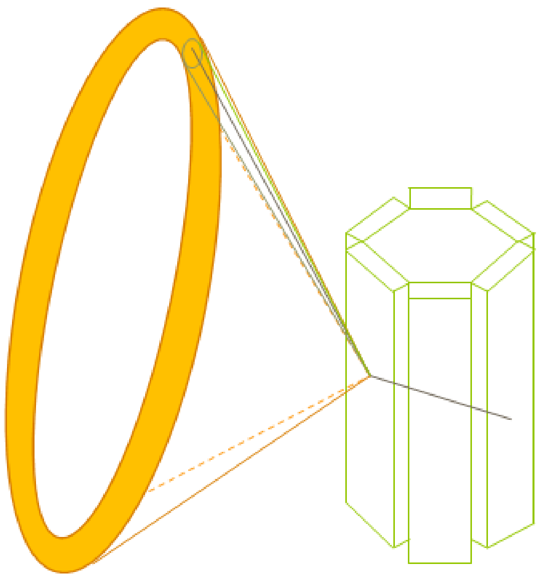

Figure 10 Compton Cone that Encompasses All Possible Directions of the Incident Photon

The thickness of the hollow Compton cone (orange cones in Figure 10) represents all possible source locations associated with the original Compton scattering angle. The use of one Compton scattering angle is not sufficient to produce a precise or accurate estimate of a source location. To increase the precision of the source location determination, more than one single Compton scattering event is required. Multiple events will create multiple Compton cones in the three dimensional model, which will overlap narrowing down the possible locations of the source as seen in Figure 11 where the dark purple rectangles represent the areas where all three Compton cones overlap. The best results are achieved through the use of multiple detector units to increase the variety of Compton cone positions relative to the source.

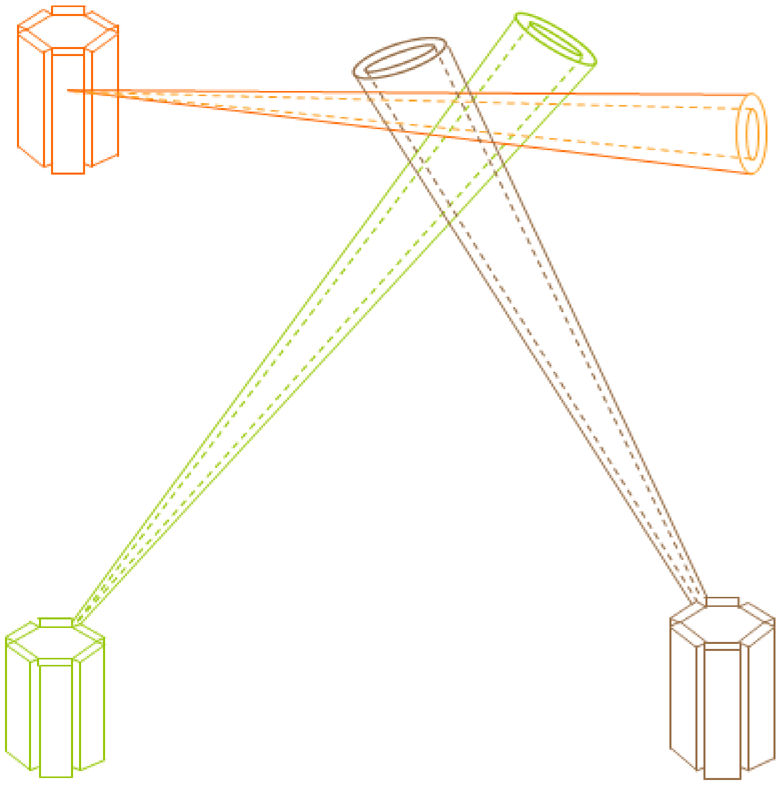

Figure 11 Multiple Detector Unit Compton Cones Narrowing Down Source Location through Overlapping Regions

The use of multiple detector units is the most efficient way to quickly triangulate the location of a source through the back projection of the scattering angle. Explanations of the mathematical calculations of simple, filtered, and iterative back-projection algorithms through the transformation of Compton cones into a series of spherical harmonics in a two plane array of lanthanum (III) bromide detectors are examined in Feng, (2009) for high resolution Compton cameras. A similar method of reconstructing cone-beam images using back-projection in spherical coordinates is found in Parra, (2002). A second method for the analysis of Compton scattering data between multiple detectors is through list-mode maximum likelihood estimation, which attempts to reconstruct the source distribution with the highest likelihood of having produced the observed data as described in Lehner et al., (2004) and Wilderman et al., (1999).

The third method for back-projection of multiple passive detection systems is through location from range differences and an extension of the Taylor Series expansion as found in Friedlander, (1987). The use of an optimized detector unit as defined in this section would allow for the most precise and accurate application of the mentioned back projection and location algorithms to identify the location of a radiologic source relative to multiple detector units.

CONCLUSIONS

The optimal dimensions of the sodium iodide detectors were determined by comparing the total number of single Compton scattering events and the percentage of multiple Compton events. There was no significant difference in the percentage of multiple Compton events for detectors of thicknesses 1.25 μx (1.88 in) and greater. Conversely, as the detector thickness increased, the number of single Compton scatter events that occurred in the detector unit decreased. It was determined that the optimized detector thickness for this experiment was 1.25 μx.

Using the detector thickness of 1.25 μx, the optimal detector length was determined using the same criteria as the thickness optimization. For the detector lengths of 4 in and greater, there was no significant difference in the percentage of multiple Compton events. Furthermore, as the length of the detector increased, the number of single Compton scatter events decreased. The number of single Compton scatter events for detector lengths of 4 and 4.5 in were not significantly different from one another, but both were found to produce a higher number of single Compton scatter events than the detectors of 5 in in length and greater. Based on these criteria, it was determine that the optimal detector length for this experiment was 4 to 4.5 inches.

Future work should examine the efficiency of the application of different back-projection methods in the optimized detector unit. Once an optimized method is determined, the minimum required number of detector units needed to produce a reasonable estimate of the source location should be identified. These additional works would determine the feasibility of using this detector unit for source tracking purposes.

REFERENCES

(2015). “Arthur Compton”. American Physical Society. Retrieved from http://www.aps.org/programs/outreach/history/historicsites/compton.cfm

Cember, H., & Johnson, T. (2009). *Introduction to health physics* (4th ed.). New York: McGraw-Hill Medical.

Feng, Y. (2009). *Design and evaluation of gamma imaging systems of Compton and hybrid cameras* (Doctoral dissertation). University of Florida, Gainesville, Florida.

Friedlander, (1987). “A passive localization algorithm and its accuracy analysis”. *IEEE Journal of Oceanic Engineering, 12*(1), 234-245.

Lehner, C., Zhong, H., & Feng, Z. (2004). “4π Compton Imaging Using a 3-D Position-Sensitive CdZnTe Detector via Weighted List-Mode Maximum Likelihood”. *IEEE Transactions on Nuclear Science, 51*(4), 1618-1624

Johnson, T., & Birky, B. (2012). *Health physics and radiological health* (4th ed.). Philadelphia: Wolters Kluwer Health/Lippincott Williams & Wilkins.

Kiser, M. (2010). “OSCAR Modeling, Simulation, and Algorithm Development”. Remote Sensing Laboratory. Andrew’s Air Force Base. Washington, D.C.

Knoll, G. (2010). *Radiation detection and measurement* (4th ed.). Hoboken, N.J.: John Wiley.

Nelson, G., & Reilly, D. (n.d.). “Gamma-Ray Interactions with Matter”. Retrieved from http://www.lanl.gov/orgs/n/n1/panda/00326397.pdf

Parra, L. (2002). “Reconstruction of cone-beam projections from Compton scattered data”. *IEEE Transactions on Nuclear Science, 47*(4), 1543-1550

Suzuki, Y., Yamaguchi, M., Odaka, H., Shimada, H., Yoshida, Y., Torikai, K., ... Nakano, T. (2013). “Three-dimensional and Multienergy Gamma-ray Simultaneous Imaging by Using a Si/CdTe Compton Camera”. *Radiology*, *267*(3), 941-947

“Timing and Coincidence Counting Systems”. (n.d.). Retrieved from http://www.canberra.com/literature/fundamental-principles/pdf/Timing-Coin-Counting.pdf

Turner, J., & Downing, D. (2012). *Statistical methods in radiation physics*. Weinheim: Wiley-VCH.

Wilderman, S., Clinthorne, N., & Rogers, W. (1998). “List-mode maximum likelihood reconstruction of Compton scatter camera images in nuclear medicine”. *Nuclear Science Symposium, 1998. Conference Record., 3*, 1716-1720

APPENDIX A

Geant4 Files

Include Folder

PhysListEmStandard.hh

```
#ifndef PhysListEmStandard_h
#define PhysListEmStandard_h 1

#include "G4VPhysicsConstructor.hh"
#include "globals.hh"

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

class PhysListEmStandard : public G4VPhysicsConstructor
{
  public:
    PhysListEmStandard(const G4String& name = "standard");
   ~PhysListEmStandard();

  public:
    // This method is dummy for physics
    virtual void ConstructParticle() {};

    // This method will be invoked in the Construct() method.
    // each physics process will be instantiated and
    // registered to the process manager of each particle type
    virtual void ConstructProcess();
};

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

#endif
```

ActionInitialization.hh

```
#ifndef THActionInitialization_h
#define THActionInitialization_h 1

#include "G4VUserActionInitialization.hh"

/// Action initialization class.
///

class THActionInitialization : public G4VUserActionInitialization
{
  public:
    THActionInitialization();
    virtual ~THActionInitialization();

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

#endif
```

Analysis.hh

```
#ifndef THAnalysis_h
#define THAnalysis_h 1

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

#endif
```

Hit.hh

```
#ifndef THHit_h
#define THHit_h 1

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

/// Calorimeter hit class
///
/// It defines data members to store the the energy deposit and track lengths
/// of charged particles in a selected volume:
/// - fEdep, fTrackLength

class THHit : public G4VHit
{
  public:
    THHit();
    THHit(const THHit&);
    virtual ~THHit();

    // operators
    const THHit& operator=(const THHit&);
    G4int operator==(const THHit&) const;

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

    // methods from base class
    virtual void Draw() {}
    virtual void Print();

    // methods to handle data
    void Add(G4double de, G4double dl);

    // get methods
    G4double GetEdep() const;
    G4double GetTrackLength() const;

  private:
    G4double fEdep;        ///< Energy deposit in the sensitive volume
    G4double fTrackLength; ///< Track length in the  sensitive volume
};

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
```

```
typedef G4THitsCollection<THHit> THHitsCollection;

extern G4ThreadLocal G4Allocator<THHit>* THHitAllocator;

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

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

inline void THHit::operator delete(void *hit)
{
  if(!THHitAllocator)
    THHitAllocator = new G4Allocator<THHit>;
  THHitAllocator->FreeSingle((THHit*) hit);
}

inline void THHit::Add(G4double de, G4double dl) {
  fEdep += de;
  fTrackLength += dl;
}

inline G4double THHit::GetEdep() const {
  return fEdep;
}

inline G4double THHit::GetTrackLength() const {
  return fTrackLength;
}

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

#endif
```

PrimaryGeneratorAction.hh

```
#ifndef THPrimaryGeneratorAction_h
#define THPrimaryGeneratorAction_h 1

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

class G4ParticleGun;
class G4Event;

/// The primary generator action class with particle gum.
///
/// It defines a single particle which hits the calorimeter
/// perpendicular to the input face. The type of the particle
/// can be changed via the G4 build-in commands of G4ParticleGun class
/// (see the macros provided with this example).
```

```
class THPrimaryGeneratorAction : public G4VUserPrimaryGeneratorAction
{
public:
  THPrimaryGeneratorAction();
  virtual ~THPrimaryGeneratorAction();

  virtual void GeneratePrimaries(G4Event* event);

  const G4ParticleGun* GetParticleGun() const { return fParticleGun; }

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

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

#endif
```

RunAction.hh

```
#ifndef THRunAction_h
#define THRunAction_h 1

#include "G4UserRunAction.hh"
#include "globals.hh"
#include <map>

class THDetectorConstruction;
class THPrimaryGeneratorAction;
class G4Run;

/// Run action class
///
/// It accumulates statistic and computes dispersion of the energy deposit
/// and track lengths of charged particles with use of analysis tools:
/// H1D histograms are created in BeginOfRunAction() for the following
/// physics quantities:
/// - Edep in absorber
/// - Edep in gap
/// - Track length in absorber
/// - Track length in gap
/// The same values are also saved in the ntuple.
/// The histograms and ntuple are saved in the output file in a format
/// accoring to a selected technology in B4Analysis.hh.
///
/// In EndOfRunAction(), the accumulated statistic and computed
/// dispersion is printed.
///

class THRunAction : public G4UserRunAction
{
public:
       THRunAction(THDetectorConstruction*, THPrimaryGeneratorAction*);


  public:
    THRunAction();
    virtual ~THRunAction();

public:
    virtual void BeginOfRunAction(const G4Run*);
    virtual void   EndOfRunAction(const G4Run*);

       void CountProcesses(G4String procName) { fProcCounter[procName]++; };

private:
       THDetectorConstruction*      fDetector;
       THPrimaryGeneratorAction*    fPrimary;
       std::map<G4String, G4int>   fProcCounter;
};

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

#endif
```

SD.hh

```
#ifndef THSD_h
#define THSD_h 1

#include "G4VSensitiveDetector.hh"

#include "THHit.hh"

#include <vector>

class G4Step;
class G4HCofThisEvent;

/// Calorimeter sensitive detector class
///
/// In Initialize(), it creates one hit for each calorimeter layer and one more
/// hit for accounting the total quantities in all layers.
///
/// The values are accounted in hits in ProcessHits() function which is called
/// by Geant4 kernel at each step.

class THSD : public G4VSensitiveDetector
{
  public:
    THSD(const G4String& name,
                     const G4String& hitsCollectionName,
                     G4int nofCells);
    virtual ~THSD();

    // methods from base class
    virtual void   Initialize(G4HCofThisEvent* hitCollection);
    virtual G4bool ProcessHits(G4Step* step, G4TouchableHistory* history);
    virtual void   EndOfEvent(G4HCofThisEvent* hitCollection);

  private:
    THHitsCollection* fHitsCollection;
    G4int     fNofCells;
};
```

```
//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......

#endif
```

EventAction.hh

```
#ifndef THEventAction_h
#define THEventAction_h 1

#include "G4UserEventAction.hh"

#include "THHit.hh"

#include "globals.hh"

/// Event action class
///
/// In EndOfEventAction(), it prints the accumulated quantities of the energy
/// deposit and track lengths of charged particles in Absober and Gap layers
/// stored in the hits collections.

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

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

private:
  // methods
  THHitsCollection* GetHitsCollection(G4int hcID,
                                          const G4Event* event) const;
  void PrintEventStatistics(G4double Shape1Edep, G4double Shape1TrackLength,
         G4double Shape2Edep, G4double Shape2TrackLength,
         G4double Shape3Edep, G4double Shape3TrackLength,
         G4double Shape4Edep, G4double Shape4TrackLength,
         G4double Shape5Edep, G4double Shape5TrackLength,
         G4double Shape6Edep, G4double Shape6TrackLength) const;

  // data members
  G4int fShape1HCID;
  G4int fShape2HCID;
  G4int fShape3HCID;
  G4int fShape4HCID;
  G4int fShape5HCID;
  G4int fShape6HCID;
};

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

#endif
```

DetectorConstruction.hh

```
#ifndef THDetectorConstruction_h
#define THDetectorConstruction_h 1

#include "G4VUserDetectorConstruction.hh"
#include "globals.hh"

class G4VPhysicalVolume;
class G4GlobalMagFieldMessenger;

/// Detector construction class to define materials and geometry.
/// The calorimeter is a box made of a given number of layers. A layer consists
/// of an absorber plate and of a detection gap. The layer is replicated.
///
/// Four parameters define the geometry of the calorimeter :
///
/// - the thickness of an absorber plate,
/// - the thickness of a gap,
/// - the number of layers,
/// - the transverse size of the calorimeter (the input face is a square).
///
/// In ConstructSDandField() sensitive detectors of B4cCalorimeterSD type
/// are created and associated with the Absorber and Gap volumes.
/// In addition a transverse uniform magnetic field is defined
/// via G4GlobalMagFieldMessenger class.

class THDetectorConstruction : public G4VUserDetectorConstruction
{
  public:
    THDetectorConstruction();
    virtual ~THDetectorConstruction();

  public:
    virtual G4VPhysicalVolume* Construct();
    virtual void ConstructSDandField();

  private:
    // methods
    //
    void DefineMaterials();
    G4VPhysicalVolume* DefineVolumes();

    // data members
    //
    static G4ThreadLocal G4GlobalMagFieldMessenger*  fMagFieldMessenger;
                                      // magnetic field messenger

    G4bool  fCheckOverlaps; // option to activate checking of volumes overlaps

};

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

#endif
```

Src Folder

SteppingAction.cc

```
#include "THSteppingAction.hh"
#include "THRunAction.hh"

#include "G4RunManager.hh"

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

THSteppingAction::THSteppingAction(THRunAction* RuAct)
:G4UserSteppingAction(),fRunAction(RuAct)
{ }

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

THSteppingAction::~THSteppingAction()
{ }

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

void THSteppingAction::UserTHSteppingAction(const G4Step* aStep)
{
  G4StepPoint* endPoint = aStep->GetPostStepPoint();
  G4String procName = endPoint->GetProcessDefinedStep()->GetProcessName();

  fTHRunAction->CountProcesses(procName);

  // kill event after first interaction
  //
  G4RunManager::GetRunManager()->AbortEvent();
}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
```

SD.cc

```
#include "THSD.hh"
#include "G4HCofThisEvent.hh"
#include "G4Step.hh"
#include "G4ThreeVector.hh"
#include "G4SDManager.hh"
#include "G4ios.hh"

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

THSD::THSD(
                            const G4String& name,
                            const G4String& hitsCollectionName,
                            G4int nofCells)
 : G4VSensitiveDetector(name),
   fHitsCollection(0),
   fNofCells(nofCells)
{
  collectionName.insert(hitsCollectionName);
}

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

THSD::~THSD()
```

```
{
}

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

void THSD::Initialize(G4HCofThisEvent* hce)
{
  // Create hits collection
  fHitsCollection
    = new THHitsCollection(SensitiveDetectorName, collectionName[0]);

  // Add this collection in hce
  G4int hcID
    = G4SDManager::GetSDMpointer()->GetCollectionID(collectionName[0]);
  hce->AddHitsCollection( hcID, fHitsCollection );

  // Create hits
  // fNofCells for cells + one more for total sums
  for (G4int i=0; i<fNofCells+1; i++ ) {
    fHitsCollection->insert(new THHit());
  }
}

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

G4bool THSD::ProcessHits(G4Step* step,
                                     G4TouchableHistory*)
{
  // energy deposit
  G4double edep = step->GetTotalEnergyDeposit();

  // step length
  G4double stepLength = 0.;
  if ( step->GetTrack()->GetDefinition()->GetPDGCharge() != 0. ) {
    stepLength = step->GetStepLength();
  }

  if ( edep==0. && stepLength == 0. ) return false;

  G4TouchableHistory* touchable
    = (G4TouchableHistory*)(step->GetPreStepPoint()->GetTouchable());

  // Get calorimeter cell id
  G4int layerNumber = touchable->GetReplicaNumber(1);

  // Get hit accounting data for this cell
  THHit* hit = (*fHitsCollection)[layerNumber];
  if ( ! hit ) {
    G4ExceptionDescription msg;
    msg << "Cannot access hit " << layerNumber;
    G4Exception("B4cCalorimeterSD::ProcessHits()",
      "MyCode0004", FatalException, msg);
  }

  // Get hit for total accounting
  THHit* hitTotal
    = (*fHitsCollection)[fHitsCollection->entries()-1];
  // Add values
  hit->Add(edep, stepLength);
  hitTotal->Add(edep, stepLength);

  return true;
}

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

void THSD::EndOfEvent(G4HCofThisEvent*)
{
  if ( verboseLevel>1 ) {
     G4int nofHits = fHitsCollection->entries();
     G4cout << "\n-------->Hits Collection: in this event they are " << nofHits
            << " hits in the tracker chambers: " << G4endl;
     for ( G4int i=0; i<nofHits; i++ ) (*fHitsCollection)[i]->Print();
  }
}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
```

RunAction.cc

```
#include "THRunAction.hh"
#include "THAnalysis.hh"

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

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

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

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

  // Create directories
  //analysisManager->SetHistoDirectoryName("histograms");
  //analysisManager->SetNtupleDirectoryName("ntuple");
  analysisManager->SetVerboseLevel(1);
  analysisManager->SetFirstHistoId(1);

  // Book histograms, ntuple
  //

  // Creating histograms
  analysisManager->CreateH1("1","Edep in Shape1", 100, 0., 800*MeV);
  analysisManager->CreateH1("2", "Edep in Shape2", 100, 0., 800 * MeV);
  analysisManager->CreateH1("3", "Edep in Shape3", 100, 0., 800 * MeV);
  analysisManager->CreateH1("4", "Edep in Shape4", 100, 0., 800 * MeV);
  analysisManager->CreateH1("5", "Edep in Shape5", 100, 0., 800 * MeV);
  analysisManager->CreateH1("6", "Edep in Shape6", 100, 0., 800 * MeV);
  analysisManager->CreateH1("7","trackL in Shape1", 100, 0., 1*m);
  analysisManager->CreateH1("8", "trackL in Shape2", 100, 0., 1 * m);
  analysisManager->CreateH1("9", "trackL in Shape3", 100, 0., 1 * m);
  analysisManager->CreateH1("10", "trackL in Shape4", 100, 0., 1 * m);
  analysisManager->CreateH1("11", "trackL in Shape5", 100, 0., 1 * m);
  analysisManager->CreateH1("12", "trackL in Shape6", 100, 0., 1 * m);


  // Creating ntuple
  //
  analysisManager->CreateNtuple("B1", "Edep and TrackL");
  analysisManager->CreateNtupleDColumn("EShape1");
  analysisManager->CreateNtupleDColumn("EShape2");
  analysisManager->CreateNtupleDColumn("EShape3");
  analysisManager->CreateNtupleDColumn("EShape4");
  analysisManager->CreateNtupleDColumn("EShape5");
  analysisManager->CreateNtupleDColumn("EShape6");
  analysisManager->CreateNtupleDColumn("LShape1");
  analysisManager->CreateNtupleDColumn("LShape2");
  analysisManager->CreateNtupleDColumn("LShape3");
  analysisManager->CreateNtupleDColumn("LShape4");
  analysisManager->CreateNtupleDColumn("LShape5");
  analysisManager->CreateNtupleDColumn("LShape6");
  analysisManager->FinishNtuple();
}

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

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

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

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

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

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

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

void THRunAction::EndOfRunAction(const G4Run* /*run*/)
{
  // print histogram statistics
```

```
//
G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();
if (analysisManager->GetH1(1)) {
       G4cout << "\n ----> print histograms statistic ";
       if (isMaster) {
              G4cout << "for the entire run \n" << G4endl;
       }
       else {
              G4cout << "for the local thread \n" << G4endl;
       }

       G4cout << " EShape1 : mean = "
              << G4BestUnit(analysisManager->GetH1(1)->mean(), "Energy")
              << " rms = "
              << G4BestUnit(analysisManager->GetH1(1)->rms(), "Energy") << G4endl;

       G4cout << " EShape2 : mean = "
              << G4BestUnit(analysisManager->GetH1(2)->mean(), "Energy")
              << " rms = "
              << G4BestUnit(analysisManager->GetH1(2)->rms(), "Energy") << G4endl;

       G4cout << " EShape3 : mean = "
              << G4BestUnit(analysisManager->GetH1(3)->mean(), "Energy")
              << " rms = "
              << G4BestUnit(analysisManager->GetH1(3)->rms(), "Energy") << G4endl;

       G4cout << " EShape4 : mean = "
              << G4BestUnit(analysisManager->GetH1(4)->mean(), "Energy")
              << " rms = "
              << G4BestUnit(analysisManager->GetH1(4)->rms(), "Energy") << G4endl;

       G4cout << " EShape5 : mean = "
              << G4BestUnit(analysisManager->GetH1(5)->mean(), "Energy")
              << " rms = "
              << G4BestUnit(analysisManager->GetH1(5)->rms(), "Energy") << G4endl;

       G4cout << " EShape6 : mean = "
              << G4BestUnit(analysisManager->GetH1(6)->mean(), "Energy")
              << " rms = "
              << G4BestUnit(analysisManager->GetH1(6)->rms(), "Energy") << G4endl;

       G4cout << " LShape1 : mean = "
              << G4BestUnit(analysisManager->GetH1(7)->mean(), "Length")
              << " rms = "
              << G4BestUnit(analysisManager->GetH1(7)->rms(), "Length") << G4endl;

       G4cout << " LShape2 : mean = "
              << G4BestUnit(analysisManager->GetH1(8)->mean(), "Length")
              << " rms = "
              << G4BestUnit(analysisManager->GetH1(8)->rms(), "Length") << G4endl;

       G4cout << " LShape3 : mean = "
              << G4BestUnit(analysisManager->GetH1(9)->mean(), "Length")
              << " rms = "
              << G4BestUnit(analysisManager->GetH1(9)->rms(), "Length") << G4endl;

       G4cout << " LShape4 : mean = "
              << G4BestUnit(analysisManager->GetH1(10)->mean(), "Length")
```

```
                << " rms = "
                << G4BestUnit(analysisManager->GetH1(10)->rms(), "Length") << G4endl;

         G4cout << " LShape5 : mean = "
                << G4BestUnit(analysisManager->GetH1(11)->mean(), "Length")
                << " rms = "
                << G4BestUnit(analysisManager->GetH1(11)->rms(), "Length") << G4endl;

         G4cout << " LShape6 : mean = "
                << G4BestUnit(analysisManager->GetH1(11)->mean(), "Length")
                << " rms = "
                << G4BestUnit(analysisManager->GetH1(11)->rms(), "Length") << G4endl;
  }

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

}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
```

PrimaryGeneratorAction.cc

```
#include "THPrimaryGeneratorAction.hh"
#include "G4RunManager.hh"
#include "G4Event.hh"
#include "G4ParticleGun.hh"
#include "G4ParticleTable.hh"
#include "G4IonTable.hh"
#include "G4ParticleDefinition.hh"
#include "G4ChargedGeantino.hh"
#include "G4SystemOfUnits.hh"
#include "Randomize.hh"
#include "G4Gamma.hh"

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

THPrimaryGeneratorAction::THPrimaryGeneratorAction()
       : G4VUserPrimaryGeneratorAction(),
       fParticleGun(0)
{
       G4int n_particle = 1;
       fParticleGun = new G4ParticleGun(n_particle);

       // default particle kinematic

       G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
       G4ParticleDefinition* particle
              = particleTable->FindParticle("gamma");
       fParticleGun->SetParticleDefinition(particle);

       //
       // fixed position
       //
       G4double x0 = 0 * cm, y0 = 0 * cm;
```

```
	G4double z0 = -85* cm;
	fParticleGun->SetParticlePosition(G4ThreeVector(x0, y0, z0));
	fParticleGun->SetParticleEnergy(662. * keV);
	//The default direction is the z-axis (i.e. towards the detector).
	//However, if the primary particle is an unstable nucleus, Geant4
	//will take care of the production of the final decay state, and the
	//products will be emitted isotropically.
	fParticleGun->SetParticleMomentumDirection(G4ThreeVector(-1., 0., 1.));
}

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

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

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

void THPrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{
	G4ParticleDefinition* particle = fParticleGun->GetParticleDefinition();

	//If the primary particle is defined to be a charged geantino (default),
	//a Cs-137 nucleus is generated instead. The primary particle can be
	//overridden at run time by the command /gun/particle
	//
	if (particle == G4Gamma::Gamma()) {
		//Cs-137
		G4int Z = 55, A = 137;
		G4double ionCharge = 0.*eplus;
		G4double excitEnergy = 0.*keV;

		G4ParticleDefinition* ion
			= G4IonTable::GetIonTable()->GetIon(Z, A, excitEnergy);

		fParticleGun->SetParticleDefinition(G4Gamma::Definition());
		fParticleGun->SetParticleEnergy(662.0*keV); //at rest

		//isotropic: flat in cosTheta and phi
		//Randomize it
		G4double cosTheta = G4UniformRand(); //cosTheta in [0,1] --> theta in
[0,pi/2]
		G4double phi = G4UniformRand() * 360 * deg; //flat in [0,2pi]
		G4double sinTheta = std::sqrt(1. - cosTheta*cosTheta);

		G4ThreeVector dir(sinTheta*std::cos(phi), sinTheta*std::sin(phi),
cosTheta);
		fParticleGun->SetParticleMomentumDirection(dir);



	}
	//create vertex
	//
	fParticleGun->GeneratePrimaryVertex(anEvent);
}
//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
```

Hit.cc

```
#include "THHit.hh"
#include "G4UnitsTable.hh"
#include "G4VVisManager.hh"
#include "G4Circle.hh"
#include "G4Colour.hh"
#include "G4VisAttributes.hh"

#include <iomanip>

G4ThreadLocal G4Allocator<THHit>* THHitAllocator = 0;

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

THHit::THHit()
 : G4VHit(),
   fEdep(0.),
   fTrackLength(0.)
{}

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

THHit::~THHit() {}

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

THHit::THHit(const THHit& right)
  : G4VHit()
{
  fEdep        = right.fEdep;
  fTrackLength = right.fTrackLength;
}

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

const THHit& THHit::operator=(const THHit& right)
{
  fEdep        = right.fEdep;
  fTrackLength = right.fTrackLength;

  return *this;
}

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

G4int THHit::operator==(const THHit& right) const
{
  return ( this == &right ) ? 1 : 0;
}

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

void THHit::Print()
{
  G4cout
     << "Edep: "
     << std::setw(7) << G4BestUnit(fEdep,"Energy")
     << " track length: "
     << std::setw(7) << G4BestUnit( fTrackLength,"Length")
     << G4endl;
}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
```

EventAction.cc

```
#include "THEventAction.hh"
#include "THSD.hh"
#include "THHit.hh"
#include "THAnalysis.hh"

#include "G4RunManager.hh"
#include "G4Event.hh"
#include "G4SDManager.hh"
#include "G4HCofThisEvent.hh"
#include "G4UnitsTable.hh"

#include "Randomize.hh"
#include <iomanip>

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

THEventAction::THEventAction()
 : G4UserEventAction(),
 fShape1HCID(-1),
 fShape2HCID(-1),
 fShape3HCID(-1),
 fShape4HCID(-1),
 fShape5HCID(-1),
 fShape6HCID(-1)
{}

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

THEventAction::~THEventAction()
{}

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

THHitsCollection*
THEventAction::GetHitsCollection(G4int hcID,
                                  const G4Event* event) const
{
  THHitsCollection* hitsCollection
    = static_cast<THHitsCollection*>(
      event->GetHCofThisEvent()->GetHC(hcID));

  if ( ! hitsCollection ) {
    G4ExceptionDescription msg;
    msg << "Cannot access hitsCollection ID " << hcID;
    G4Exception("B4cEventAction::GetHitsCollection()",
      "MyCode0003", FatalException, msg);
  }
  return hitsCollection;
}

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

void THEventAction::PrintEventStatistics(
       G4double Shape1Edep, G4double Shape1TrackLength,
       G4double Shape2Edep, G4double Shape2TrackLength,
       G4double Shape3Edep, G4double Shape3TrackLength,
       G4double Shape4Edep, G4double Shape4TrackLength,
       G4double Shape5Edep, G4double Shape5TrackLength,
       G4double Shape6Edep, G4double Shape6TrackLength) const
{
  // print event statistics
       G4cout
              << "   Shape1: total energy: "
              << std::setw(7) << G4BestUnit(Shape1Edep, "Energy")
              << "       total track length: "
              << std::setw(7) << G4BestUnit(Shape1TrackLength, "Length")
              << G4endl
              << "        Shape2: total energy: "
              << std::setw(7) << G4BestUnit(Shape2Edep, "Energy")
              << "       total track length: "
              << std::setw(7) << G4BestUnit(Shape1TrackLength, "Length")
              << G4endl
              << "        Shape3: total energy: "
              << std::setw(7) << G4BestUnit(Shape3Edep, "Energy")
              << "       total track length: "
              << std::setw(7) << G4BestUnit(Shape3TrackLength, "Length")
              << G4endl
              << "        Shape4: total energy: "
              << std::setw(7) << G4BestUnit(Shape4Edep, "Energy")
              << "       total track length: "
              << std::setw(7) << G4BestUnit(Shape4TrackLength, "Length")
              << G4endl
              << "        Shape5: total energy: "
              << std::setw(7) << G4BestUnit(Shape5Edep, "Energy")
              << "       total track length: "
              << std::setw(7) << G4BestUnit(Shape5TrackLength, "Length")
              << G4endl
              << "        Shape6: total energy: "
              << std::setw(7) << G4BestUnit(Shape6Edep, "Energy")
              << "       total track length: "
              << std::setw(7) << G4BestUnit(Shape6TrackLength, "Length")
              << G4endl;
}

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

void THEventAction::BeginOfEventAction(const G4Event* /*event*/)
{}

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

void THEventAction::EndOfEventAction(const G4Event* event)
{
  // Get hits collections IDs (only once)
```

```
      if (fShape1HCID == -1) {
             fShape1HCID
                    = G4SDManager::GetSDMpointer()-
>GetCollectionID("Shape1HitsCollection");
             fShape2HCID
                    = G4SDManager::GetSDMpointer()-
>GetCollectionID("Shape2HitsCollection");
             fShape3HCID
                    = G4SDManager::GetSDMpointer()-
>GetCollectionID("Shape3HitsCollection");
             fShape4HCID
                    = G4SDManager::GetSDMpointer()-
>GetCollectionID("Shape4HitsCollection");
             fShape5HCID
                    = G4SDManager::GetSDMpointer()-
>GetCollectionID("Shape5HitsCollection");
             fShape6HCID
                    = G4SDManager::GetSDMpointer()-
>GetCollectionID("Shape6HitsCollection");
  }

  // Get hits collections
      THHitsCollection* Shape1HC = GetHitsCollection(fShape1HCID, event);
      THHitsCollection* Shape2HC = GetHitsCollection(fShape2HCID, event);
      THHitsCollection* Shape3HC = GetHitsCollection(fShape3HCID, event);
      THHitsCollection* Shape4HC = GetHitsCollection(fShape4HCID, event);
      THHitsCollection* Shape5HC = GetHitsCollection(fShape5HCID, event);
      THHitsCollection* Shape6HC = GetHitsCollection(fShape6HCID, event);

  // Get hit with total values
      THHit* Shape1Hit = (*Shape1HC)[Shape1HC->entries() - 1];
      THHit* Shape2Hit = (*Shape2HC)[Shape1HC->entries() - 1];
      THHit* Shape3Hit = (*Shape3HC)[Shape1HC->entries() - 1];
      THHit* Shape4Hit = (*Shape4HC)[Shape1HC->entries() - 1];
      THHit* Shape5Hit = (*Shape5HC)[Shape1HC->entries() - 1];
      THHit* Shape6Hit = (*Shape6HC)[Shape1HC->entries() - 1];

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

    PrintEventStatistics(
             Shape1Hit->GetEdep(), Shape1Hit->GetTrackLength(),
             Shape2Hit->GetEdep(), Shape2Hit->GetTrackLength(),
             Shape3Hit->GetEdep(), Shape3Hit->GetTrackLength(),
             Shape4Hit->GetEdep(), Shape4Hit->GetTrackLength(),
             Shape5Hit->GetEdep(), Shape5Hit->GetTrackLength(),
             Shape6Hit->GetEdep(), Shape6Hit->GetTrackLength());
  }

  // Fill histograms, ntuple
  //

  // get analysis manager
  G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();
  // fill histograms
  analysisManager->FillH1(1, Shape1Hit->GetEdep());
  analysisManager->FillH1(2, Shape2Hit->GetEdep());
  analysisManager->FillH1(3, Shape3Hit->GetEdep());
  analysisManager->FillH1(4, Shape4Hit->GetEdep());
  analysisManager->FillH1(5, Shape5Hit->GetEdep());
  analysisManager->FillH1(6, Shape6Hit->GetEdep());
  analysisManager->FillH1(7, Shape1Hit->GetTrackLength());
  analysisManager->FillH1(8, Shape2Hit->GetTrackLength());
  analysisManager->FillH1(9, Shape3Hit->GetTrackLength());
  analysisManager->FillH1(10, Shape4Hit->GetTrackLength());
  analysisManager->FillH1(11, Shape5Hit->GetTrackLength());
  analysisManager->FillH1(12, Shape6Hit->GetTrackLength());

  // fill ntuple
  analysisManager->FillNtupleDColumn(0, Shape1Hit->GetEdep());
  analysisManager->FillNtupleDColumn(1, Shape2Hit->GetEdep());
  analysisManager->FillNtupleDColumn(2, Shape3Hit->GetEdep());
  analysisManager->FillNtupleDColumn(3, Shape4Hit->GetEdep());
  analysisManager->FillNtupleDColumn(4, Shape5Hit->GetEdep());
  analysisManager->FillNtupleDColumn(5, Shape6Hit->GetEdep());
  analysisManager->FillNtupleDColumn(6, Shape1Hit->GetTrackLength());
  analysisManager->FillNtupleDColumn(7, Shape2Hit->GetTrackLength());
  analysisManager->FillNtupleDColumn(8, Shape3Hit->GetTrackLength());
  analysisManager->FillNtupleDColumn(9, Shape4Hit->GetTrackLength());
  analysisManager->FillNtupleDColumn(10, Shape5Hit->GetTrackLength());
  analysisManager->FillNtupleDColumn(11, Shape6Hit->GetTrackLength());
  analysisManager->AddNtupleRow();
}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
```

DetectorConstruction.cc

```
#include "THDetectorConstruction.hh"
#include "THSD.hh"
#include "G4Material.hh"
#include "G4NistManager.hh"

#include "G4Box.hh"
#include "G4LogicalVolume.hh"
#include "G4PVPlacement.hh"
#include "G4PVReplica.hh"
#include "G4GlobalMagFieldMessenger.hh"
#include "G4AutoDelete.hh"

#include "G4SDManager.hh"

#include "G4VisAttributes.hh"
#include "G4Colour.hh"

#include "G4PhysicalConstants.hh"
#include "G4SystemOfUnits.hh"

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
```

```
G4ThreadLocal
G4GlobalMagFieldMessenger* THDetectorConstruction::fMagFieldMessenger = 0;

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

THDetectorConstruction::THDetectorConstruction()
 : G4VUserDetectorConstruction(),
   fCheckOverlaps(true)

{
}

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

THDetectorConstruction::~THDetectorConstruction()
{
}

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

G4VPhysicalVolume* THDetectorConstruction::Construct()
{
  // Define materials
  DefineMaterials();

  // Define volumes
  return DefineVolumes();
}

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

void THDetectorConstruction::DefineMaterials()
{
       // Get nist material manager
       G4NistManager* nist = G4NistManager::Instance();

       G4Material* world_mat = nist->FindOrBuildMaterial("G4_AIR");

       G4Material* NaI = nist->FindOrBuildMaterial("G4_SODIUM_IODIDE");

  // Print materials
  G4cout << *(G4Material::GetMaterialTable()) << G4endl;
}

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

G4VPhysicalVolume* THDetectorConstruction::DefineVolumes()
{
       G4bool checkOverlaps = true;
  //
  // World
  //
       G4NistManager* nist = G4NistManager::Instance();
       G4Material* world_mat = nist->FindOrBuildMaterial("G4_AIR");
       G4VSolid* solidWorld =
              new G4Box("World",        // its name
              10.0*m, 10.0*m, 10.0*m);
```

```
        G4LogicalVolume* logicWorld =
               new G4LogicalVolume(solidWorld,     //its solid
               world_mat,      // its material
               "World");       // its name

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

        //Shape 1
        //


        G4Material* NaI = nist->FindOrBuildMaterial("G4_SODIUM_IODIDE");
        G4ThreeVector pos1 = G4ThreeVector(-7.25*2.54*cm, 0, -4.25 * 2.54*cm);

        G4RotationMatrix*yRot = new G4RotationMatrix;  //its rotation
        yRot->rotateY(30.*deg);

        G4VSolid* pBoxSolid = new G4Box("Shape 1", 1.5*2.54*cm, 16.0*2.54*cm, 4*2.54*cm);
//Box shape


        G4LogicalVolume* pBoxLog =
               new G4LogicalVolume(pBoxSolid,        //its solid
               NaI,              //its material
               "Shape1");        //its name

        new G4PVPlacement(yRot,                   //its rotation
               pos1,                 //at position
               pBoxLog,              //its logical volume
               "Shape1",             //its name
               logicWorld,           //its mother volume
               false,                //no boolean operation
               0,                    //copy number
               checkOverlaps);       //overlaps checking




        //
        //Shape 2
        //
        G4ThreeVector pos2 = G4ThreeVector(-7.25 * 2.54*cm, 0, 4.5* 2.54*cm);

        G4VSolid* pBoxSolid2 = new G4Box("Shape 2", 1.5*2.54*cm, 16.0*2.54*cm,4*2.54*cm);

        G4RotationMatrix*yRot2 = new G4RotationMatrix;   //its rotation
        yRot2->rotateY(330.*deg);

        G4LogicalVolume* pBoxLog2 =
               new G4LogicalVolume(pBoxSolid2,              //its solid
               NaI,                                            //its material
               "Shape2");                                      //its name

        new G4PVPlacement(yRot2,                          //its rotation
               pos2,                                         //at position
               pBoxLog2,                                     //its logical volume
               "Shape2",                                     //its name
               logicWorld,                                  //its mother volume
               false,                                        //no boolean operation
               0,                                           //copy number
               checkOverlaps);                              //overlaps checking


        //
        //Shape 3
        //
        G4ThreeVector pos3 = G4ThreeVector(0, 0, 8.75 * 2.54*cm);

        G4VSolid* pBoxSolid3 = new G4Box("Shape 3", 4*2.54*cm, 16.0*2.54*cm, 1.5*2.54*cm);

        G4LogicalVolume* pBoxLog3 =
               new G4LogicalVolume(pBoxSolid3,         //its solid
               NaI,                                    //its material
               "Shape3");                              //its name

        new G4PVPlacement(0,                      //no rotation
               pos3,                                 //its position
               pBoxLog3,                             //its logical volume
               "Shape3",                             //its name
               logicWorld,                           //its mother volume
               false,                                //no boolean operation
               0,                                    //its copy number
               checkOverlaps);                       //overlaps chekcing


        //
        //Shape 4
        //
        G4ThreeVector pos4 = G4ThreeVector(7.25 * 2.54*cm, 0, 4.5* 2.54*cm);

        G4VSolid* pBoxSolid4 = new G4Box("Shape 4", 1.5*2.54*cm, 16.0*2.54*cm, 4*2.54*cm);


        G4LogicalVolume* pBoxLog4 =
               new G4LogicalVolume(pBoxSolid4,         //its solid
               NaI,                                     //its material
               "Shape4");                              //its name

        new G4PVPlacement(yRot,                     //its rotation
               pos4,                                   //its position
               pBoxLog4,                               //its logical volume
               "Shape4",                               //its name
               logicWorld,                             //its mother volume
               false,                                  //no boolean operation
               0,                                      //copy number
               checkOverlaps);                         //overlaps checking
```

```
     //
     //Shape 5
     //
     G4ThreeVector pos5 = G4ThreeVector(7.25 * 2.54*cm, 0, -4.25 * 2.54*cm);

     G4VSolid* pBoxSolid5 = new G4Box("Shape 5", 1.5*2.54*cm, 16.0*2.54*cm, 4*2.54*cm);


     G4LogicalVolume* pBoxLog5 =
            new G4LogicalVolume(pBoxSolid5,      //its solid
            NaI,                                 //its material
            "Shape5");                           //its name

     new G4PVPlacement(yRot2,                 //its rotation
            pos5,                                //its position
            pBoxLog5,                            //its logical volume
            "Shape5",                            //its name
            logicWorld,                          //its mother volume
            false,                                //no boolean operation
            0,                                   //copy number
            checkOverlaps);                       //overlaps checking

     //
     //Shape 6
     //
     G4ThreeVector pos6 = G4ThreeVector(0, 0, -8.5 * 2.54*cm);

     G4VSolid* pBoxSolid6 = new G4Box("Shape 6", 4*2.54*cm, 16.0*2.54*cm, 1.5*2.54*cm);


     G4LogicalVolume* pBoxLog6 =
            new G4LogicalVolume(pBoxSolid6,      //its solid
            NaI,                                 //its material
            "Shape6");                           //its name

     new G4PVPlacement(0,                     //no rotation
            pos6,                                //its position
            pBoxLog6,                            //its logical volume
            "Shape6",                            //its name
            logicWorld,                          //its mother volume
            false,                               //no boolean operation
            0,                                   //copy number
            checkOverlaps);                      //overlaps checking



//
// Visualization attributes
//
     G4VisAttributes* visAttributes = new G4VisAttributes(G4Colour(1.0, 1.0, 1.0));
     visAttributes->SetVisibility(false);
     logicWorld->SetVisAttributes(visAttributes);

visAttributes = new G4VisAttributes(G4Colour(1.0, 0.0, 0.0));  //red
pBoxLog->SetVisAttributes(visAttributes);

visAttributes = new G4VisAttributes(G4Colour(0.0, 1.0, 0.0));  //green
pBoxLog2->SetVisAttributes(visAttributes);
```

```
  visAttributes = new G4VisAttributes(G4Colour(0.0, 0.0, 1.0));  //ble
  pBoxLog3->SetVisAttributes(visAttributes);

  visAttributes = new G4VisAttributes(G4Colour(0.0, 1.0, 1.0)); //cyan
  pBoxLog4->SetVisAttributes(visAttributes);

  visAttributes = new G4VisAttributes(G4Colour(1.0, 0.0, 1.0)); //magenta
  pBoxLog5->SetVisAttributes(visAttributes);

  visAttributes = new G4VisAttributes(G4Colour(1.0, 1.0, 0.0)); //yellow
  pBoxLog6->SetVisAttributes(visAttributes);


  //
  // Always return the physical World
  //
  return physWorld;
}

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

void THDetectorConstruction::ConstructSDandField()
{
  // G4SDManager::GetSDMpointer()->SetVerboseLevel(1);

  //
  // Sensitive detectors
  //

       THSD* Shape1SD
              = new THSD("Shape1SD", "Shape1HitsCollection", 0);
       SetSensitiveDetector("Shape1", Shape1SD);

       THSD* Shape2SD
              = new THSD("Shape2SD", "Shape2HitsCollection", 0);
       SetSensitiveDetector("Shape2", Shape2SD);

       THSD* Shape3SD
              = new THSD("Shape3SD", "Shape3HitsCollection", 0);
       SetSensitiveDetector("Shape3", Shape3SD);

       THSD* Shape4SD
              = new THSD("Shape4SD", "Shape4HitsCollection", 0);
       SetSensitiveDetector("Shape4", Shape4SD);

       THSD* Shape5SD
              = new THSD("Shape5SD", "Shape5HitsCollection", 0);
       SetSensitiveDetector("Shape5", Shape5SD);

       THSD* Shape6SD
              = new THSD("Shape6SD", "Shape6HitsCollection", 0);
       SetSensitiveDetector("Shape6", Shape6SD);
}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
```

ActionInitialization.cc

```
#include "THActionInitialization.hh"
#include "THPrimaryGeneratorAction.hh"
#include "THRunAction.hh"
#include "THEventAction.hh"

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

THActionInitialization::THActionInitialization()
 : G4VUserActionInitialization()
{}

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

THActionInitialization::~THActionInitialization()
{}

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

void THActionInitialization::BuildForMaster() const
{
  SetUserAction(new THRunAction);
}

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

void THActionInitialization::Build() const
{
  SetUserAction(new THPrimaryGeneratorAction);
  SetUserAction(new THRunAction);
  SetUserAction(new THEventAction);
}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
```

PhysListEmStandard.cc

```
#include "PhysListEmStandard.hh"
#include "G4ParticleDefinition.hh"
#include "G4ProcessManager.hh"
#include "G4PhysicsListHelper.hh"

#include "G4ComptonScattering.hh"
#include "G4GammaConversion.hh"
#include "G4PhotoElectricEffect.hh"

#include "G4SystemOfUnits.hh"

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

PhysListEmStandard::PhysListEmStandard(const G4String& name)
   :  G4VPhysicsConstructor(name)
{}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
```

```
PhysListEmStandard::~PhysListEmStandard()
{}

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

void PhysListEmStandard::ConstructProcess()
{
  G4PhysicsListHelper* ph = G4PhysicsListHelper::GetPhysicsListHelper();

  // Add standard EM Processes
  //
  aParticleIterator->reset();
  while ((*aParticleIterator)()){
         G4ParticleDefinition* particle = aParticleIterator->value();
         G4String particleName = particle->GetParticleName();

         if (particleName == "gamma") {

                ////ph->RegisterProcess(new G4RayleighScattering, particle);
                ph->RegisterProcess(new G4PhotoElectricEffect, particle);
                ph->RegisterProcess(new G4ComptonScattering, particle);
                ph->RegisterProcess(new G4GammaConversion, particle);
         }
         }

 }
//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
```