Investigating Performance of a Scintillation Radiation Detector Design

by

Nathan J. Murtha

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE

in

Physics and Mathematics

(Department of Physics and Astronomy, Dr. A.J. Sarty supervising faculty)

....

....

....

....

....

SAINT MARY'S UNIVERSITY

May 25, 2015

Abstract

Investigating Performance of a Scintillation Radiation Detector Design, by *Nathan J. Murtha*, submitted on May 25, 2015:

The Super BigBite Spectrometer project in Hall A of the Thomas Jefferson National Accelerator Facility (JLab) is finalizing the design of a required radiation detector called the Coordinate Detector (CDet) for use in electro-nuclear scattering experiments. The CDet will be composed of two planes of scintillating bars (extruded at Fermilab) containing wavelength shifting (WLS) fibers that absorb scintillation light and transport it to photomultiplier tubes (PMTs) for detection. This thesis reports on measurements made on sample scintillating bars to assist in final decisions related to the detector design. Bars with two surface finishes were tested: (1) a TiO_2 coating applied during extrusion; and (2) a machined-flat surface subsequently wrapped with $\frac{1}{4}$ mm thick aluminized Mylar. Each bar contained two 1 mm WLS fibers. Measurements were made to compare scintillation light output of these two finishes. Longitudinal and transverse measurements were made to determine PMT photoelectron yield dependence. It was found that the TiO_2 coating and $\frac{1}{4}$ thick aluminized Mylar wrapping performed the same within experimental uncertainty. Longitudinal dependence was consistent with expectations from attenuation in the WLS fibres, while transverse dependence shows a maximum yield near the center of the bar and drop-off closer to the edges.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Modern subatomic physics makes use of particle detectors (otherwise known as radiation detectors) to detect reaction products produced in scattering experiments. Accurate detection of these reaction products is crucial, and as such there is constantly work being done to improve existing detector systems or to implement new detector systems. Hall A at the Thomas Jefferson National Accelerator Facility (JLab), located in Newport News, VA, is currently in the process of implementing an improved detector system following an upgrade to a 12 GeV electron beam.

1.1 Hall A at JLab: The Super BigBite Spectrometer and the Coordinate Detector

Experimental Hall A at JLab is currently in the process of making improvements to their suite of detectors, creating the Super BigBite Spectrometer (SBS) facility, following an upgrade in maximum electron beam energy from 6 GeV to 12 GeV. An experiment scheduled to be run in Hall A in the near future, "Large Acceptance Proton Form Factor Ratio Measurements at 13 to 15 (GeV/c^2)" (E12-07-109), will make use of this new SBS facility [1] [2].

Part of the SBS project includes the design, testing, and construction of a hodoscope Coordinate Detector (CDet) for measurement of the vertical position of scattered electrons. While the electron beam is running, there are many by-product particles (or "radiation") created from secondary reactions. Angular correlation between the elastically scattered electron and recoil proton allow the vertical electron position to be used to identify the desired proton track, even in a large flux of background particles, thereby reducing the stress on the detector triggering system. The accurate vertical position measurements of the scattered electron trajectories (resolution of 1.3 mm is expected [3]) will allow a coplanarity requirement to be implemented into the SBS triggering system (see Figure 1.1), allowing efficient detection capabilities at high background rates of up to $8{\times}10^{38}$ Hz/cm^2 [3]. The CDet will be composed

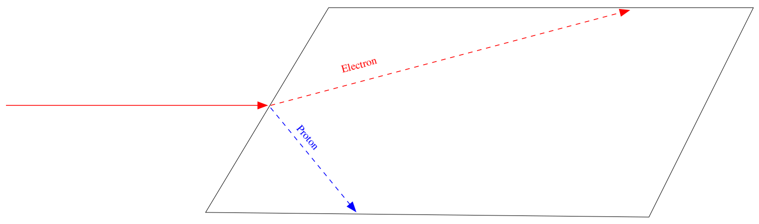
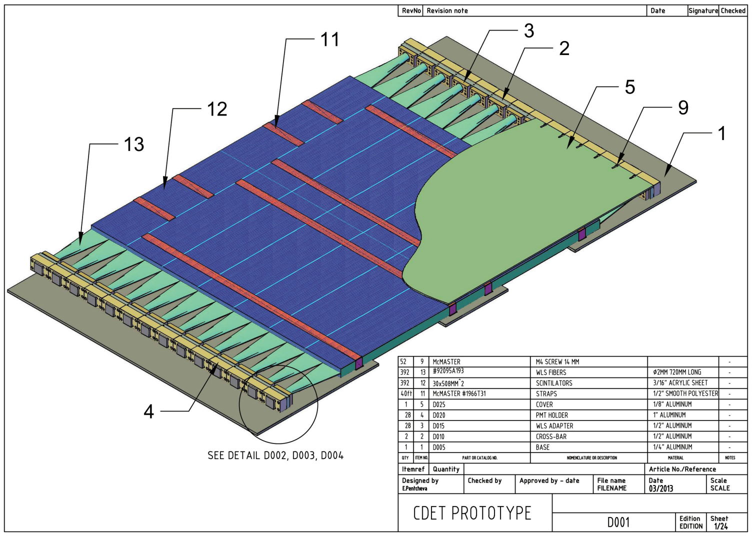


Figure 1.1: Coplanar elastic scattering of electron and proton.

of two back-to-back planes of plastic scintillating bars, each with an active area of 102 cm × 294 cm [3]. The entire CDet will contain 2352 scintillating bars, with each plane composed of 1176 scintillating bars measuring 0.5 cm tall by 51 cm long by 4 cm wide[1]. In each plane, the bars are arranged into three interlocking modules of 392 bars (2 bars across), absent of dead zones [3] (see see Figure 1.2). Along the length of each bar will be a hole, inside of which a wavelength shifting (WLS) fibre will be placed that will transmit scintillation light to a photomultiplier tube (PMT) for data collection. A 15 cm polyethylene block will be placed immediately in front of the CDet to act as a shield for low energy background radiation. In order to reduce the

[1]The original proposal for the CDet called for 3 cm wide scintillating bars, but as a result of the work presented in this thesis they were made wider.



QTY	ITEM NO.	PART OR CATALOG NO.	NOMENCLATURE OR DESCRIPTION	MATERIAL	NOTES
52	9	McMASTER	M4 SCREW 14 MM		-
392	13	#92095A193	WLS FIBERS	ø2MM 720MM LONG	-
392	12	30x508MM^2	SCINTILATORS	3/16" ACRYLIC SHEET	-
40ft	11	McMASTER #1966T31	STRAPS	1/2" SMOOTH POLYESTER	-
1	5	D025	COVER	1/8" ALUMINUM	-
28	4	D020	PMT HOLDER	1" ALUMINUM	-
28	3	D015	WLS ADAPTER	1/2" ALUMINUM	-
2	2	D010	CROSS-BAR	1/2" ALUMINUM	-
1	1	D005	BASE	1/4" ALUMINUM	-

Figure 1.2: One module in a CDet plane.

number of two-bar events, as well as to improve discrimination between the desired high energy elastically scattered electrons and background hits, the scintillator bars will be tilted along the length of the CDet following guidance provided by a GEANT3 simulation of the CDet [3] (see Figure 1.3; the z-dimensions have been scaled by a factor of 10 to make the tilt evident). This tilt is designed so that trajectories of electron tracks originating from the target-centre pass through the full 4 cm width of the scintillator bar cross sections, and reaches a maximum of 16° at the extreme ends of the CDet. The scintillator bars that will be used in the CDet will be extruded at Fermilab. The original design called for the bars to be extruded with a trapezoidal cross section, with a width of 3 cm and heights of 0.5 cm and 0.503 cm; this would have provided the required tilt of up to 16° in the scintillator bars along the length of the CDet. However, the extrusion process was found to be insufficiently uniform across the length of the scintillating bars to accomplish the construction of such a cross section [4] [5] (see Figure 1.4), having an extrusion width stability of 0.005 cm. In addition, a reflective envelope surrounding each scintillator bar is required in order

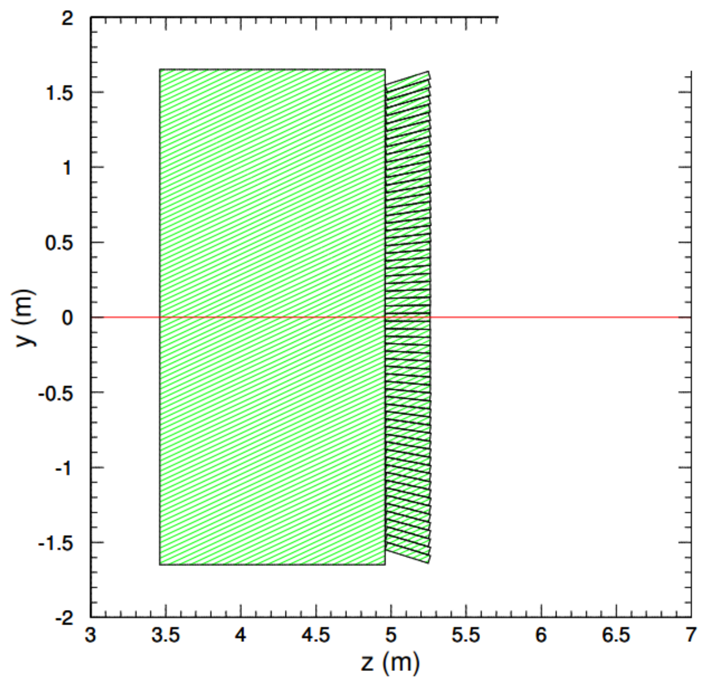


Figure 1.3: Tilting of scintillators behind polyethylene in GEANT3 geometry.

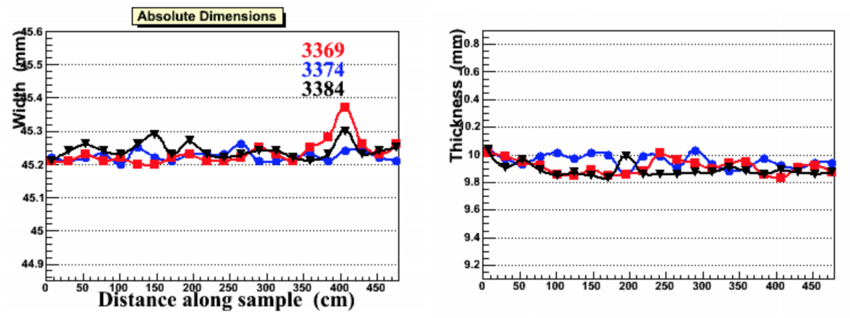


Figure 1.4: Dimensions of extruded bar sample measured along thickness and width.

to prevent scintillation light created in one bar from inadvertently passing between bars, as well as to increase scintillation light collection efficiency of the WLS fibres by increasing the reflectance of the light at the bar surface. A TiO_2 coating was initially proposed for such a role; however, the application of a TiO_2 coating only has a stability in thickness of 0.012 cm [4], which will not allow for accurate construction of the tilted scintillator bar design proposed for the CDet.

In order to remedy the problems with extrusion and reflective coating thickness stabilities, it has been proposed that the bars will extruded *without* the application of a reflective coating and precisely machined to a rectangular cross section, after which they will be wrapped with 0.25 mm thick aluminized Mylar [5]. The tilt in CDet scintillator bar elements will then be accomplished with careful use of thin shims. The extrusion process will include the extrusion of a 3 mm hole along the centre of the length of the bar, inside which a 2 mm diameter WLS fibre will be placed for light collection.

1.1.1 General Thesis Goals

The general goal of this work is to assess the performance of the proposed fix for the scintillator bar elements of the CDet. These results will be part of Saint Mary's University's contribution to the development of the CDet, and ultimately to the development of the SBS system at JLab.

Development of appropriate experimental techniques for assessment of relative detection performance of a CDet scintillator bar element is outlined herein.

1.2 Scintillating Materials

Scintillating materials are amongst the most widely used tools in nuclear physics. When ionizing radiation interacts with a scintillating material, it deposits some energy, which is converted into a small flash of light (called scintillation light). These brief flashes of light may then be detected, providing data regarding the radiation that initially interacted with the scintillating material.

Three of the key features of a general scintillator are [6]:

- Sensitive to incident radiation energy
- Fast time response
- Light emission pulse discrimination

Above a specific minimum energy threshold, scintillator light output is nearly directly proportional to the energy deposited in the scintillator. This is useful for rough estimates of the energy of incident radiation for radiation that stops within the scintillator, or for estimates of the rate of energy-loss per unit length for radiation that passes through the scintillator. In addition, scintillators have short response and recovery times (on the order of nanoseconds), allowing for good time resolution between events as well as making them well suited for high flux environments (as will be the case for the CDet). Finally, some types of scintillating material respond in specific and characteristic ways to different kinds of incident radiation, and thus the shape of the light pulses produced in the interactions can be analysed to determine what kind of radiation the signal was produced by.

There are many different kinds of scintillating detector materials: organic scintillators, inorganic crystals, gaseous scintillators, and glasses [6]. Each have their own advantages and disadvantages; the CDet will be composed of plastic scintillator bars, which are a type of organic scintillators.

1.2.1 Plastic Scintillators for Use in CDet

The CDet will be composed of plastic scintillator bars. Plastic scintillators are organic materials; they're comprised of a solution of organic scintillating material suspended in a solid plastic solvent. The organic scintillating material is an aromatic hydrocarbon compound, containing linked or condensed benzene-ring structures [6].

Scintillation light in organic scintillators, such as plastics, is produced from the excitation of free valence electrons in the scintillator molecules. These valence electrons belong to π-molecular orbitals in the scintillator molecules, and are not associated with any one atom in the molecule. Incident ionizing radiation imparts energy to these delocalized valence electrons, exciting them into higher energy states within the molecule. The ground state energy within an organic scintillator molecule is a singlet state, S_0 (see Figure 1.5). Excitation of the electron may put it into an excited singlet state (S^*, S^{**} and so on) or into a triplet state (T_0, T^*, T^{**} and so on). Due to molecular fine structure, there are excited vibrational modes associated with each energy level [6]. The energy from incident ionizing radiation excites the delocalized valence

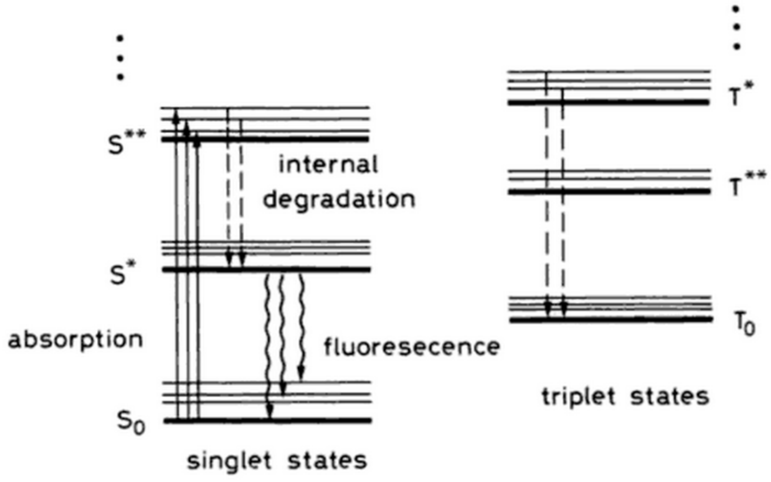


Figure 1.5: Energy levels within an organic scintillator molecule [6]. Thin lines represent excited vibrational states.

electrons into excited electron and vibrational states (indicated by the solid arrows in Figure 1.5). States excited past the S^* state in the singlet states will rapidly decay down to the S^* state within 10 ps via internal degradation (dashed arrows in Figure 1.5); no scintillation light is emitted during this process [6]. From the S^* state, the electrons will typically decay to a vibrational state of S_0 within a few nanoseconds, emitting scintillation light. Within the excited triplet states, the electron will decay to T_0 via internal degradation. T_0 states decay by interaction with another electron in a T_0 state [6]:

$$T_0 + T_0 \rightarrow S^* + S_0 + \text{phonons}$$

After which scintillation light is produced by the decay of the S^* state. Scintillation light is emitted isotropically once the decay occurs.

The plastic scintillators that will be used in the CDet are identical in composition to the Bicron BC-408 plastic scintillator. This scintillator is well suited for detection of charged particles [7], including high energy electrons. This polyvinyltoluene based scintillator has a peak emission wavelength of 425 nm.

1.3 WLS Materials

WLS materials are materials which capture light at one frequency and emit light at a lower frequency [6]. This is useful if photosensor used with a detector is not sensitive to one region of the electromagnetic spectrum; for example, scintillation light may be emitted in the blue region of the spectrum, whereas a PMT may be sensitive to the green region of the spectrum (as is the case for the CDet system). WLS materials allow for potential inefficiencies associated with potential mismatch of peak emission wavelength and peak sensitivity wavelength to be corrected for.

1.3.1 WLS in CDet

The extrusion process for the scintillator bars will include a 3 mm diameter hole, inside of which a 2 mm diameter single-clad BCF-92 WLS fibre will be inserted for light collection. The WLS fibre will be seated in the hole without any optical coupling. The BCF-92 WLS fibre has a polystyrene core with a polymethylmethacrylate cladding. The peak emission wavelength of the BCF-92 WLS fibre is 492 nm [8].

1.4 Photomultiplier Tubes

PMT's are very sensitive photon detectors that turn small amounts of light into a measurable electric current [6]. They are frequently used to collect scintillation light produced in a scintillator. In the CDet, WLS fibres transmit scintillation light to the face of a PMT for collection.

1.4.1 Operation of a PMT

A PMT is a vacuum tube consisting of a photosensitive cathode, an electron multiplier system and a terminating anode from which a measurable signal may be taken [1] [6]. A diagram of a general PMT is shown in Figure 1.6. Incident photons are directed (using optics such as a WLS fibre) onto the photocathode, which is coated with a photosensitive material having a low work function. When the photon interacts with the photocathode, an electron is released by the photoelectric effect [6]. The emitted electrons pass through a focusing electrode, which direct them onto a series of dynodes. There is a voltage difference applied between the dynodes, increasing

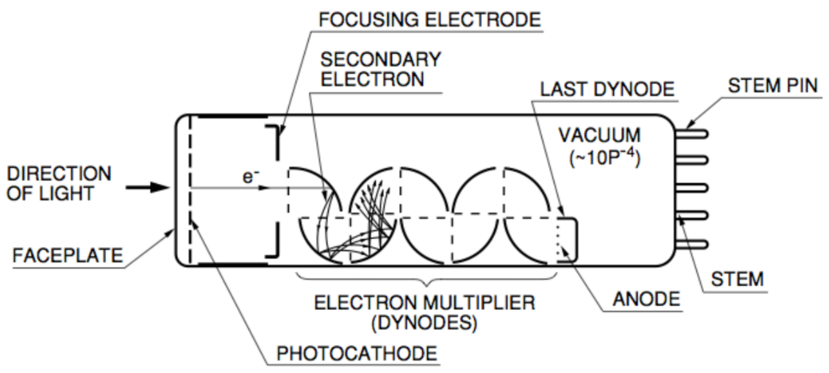


Figure 1.6: Simplified PMT schematic [1].

from one dynode to the next. When the electrons strike a dynode, they impart some energy into the material, which causes more electrons to be released; due to the applied voltage between dynodes, the electrons accelerate from the photocathode, down the series of dynodes, creating a cascade of electrons as they do so. This cascade eventually terminates at an anode, where the build-up of charge is read-out and sent to a data-acquisition system. This charge is proportional to the number of photons incident on the PMT [1] [11], allowing for assessment of the intensity of the original light pulse. The gain of a PMT determines how large the signal at the anode will be. Since plastic scintillators produce light pulses proportional to the energy deposited in them, PMTs can then be used to estimate the energy deposited by the radiation incident on the scintillator bar elements of the CDet.

1.4.2 Gain

The gain of a PMT is the ratio of the anode current to the photocathode current [11]; it is a measure of the ability of the PMT to amplify an incident light pulse to a measurable electric signal. The overall gain of a PMT depends exponentially on the number of dynodes in the dynode chain; at each dynode, an average of three to four electrons are liberated for each each incident electron [1] (see Figure 1.6).

1.4.3 Photocathode and Photoelectrons

The photocathode is responsible for starting the process of converting incident light to a detectable electric signal. The photocathode structure is coated with a thin layer of photosensitive material; some of the most common materials used include Cs-Te, Cs-I, Ag-O-Cs, and Bialkali. The PMTs used in the CDet will have Bialkali photocathodes, consisting of Sb-Ru-Cs and Sb-K-Cs [1]. When an incident photon interacts with the material of the photocathode, an electron is released by the photoelectric effect; if many photons interact with the photocathode, many electrons are released, producing a proportionally larger signal at the anode.

Electrons released from the photocathode are termed *photoelectrons*. It is these photoelectrons that pass through the dynode chain of the PMT, causing the cascade of electrons measured at the anode. A typical light pulse reaching the photocathode of a PMT will consist of many photons, and as a result the charge measured at the anode of the PMT will have been caused by many photoelectrons.

Due to the low work function of the photocathode material, single photoelectrons are emitted from the photocathode at room temperature by a process known as thermionic emission [1] [11]. If the average signal produced by a single photoelectron is known, it is possible to determine the number of photoelectrons causing any signal from the PMT. Because the gain of the PMT is independent of the initial number of photoelectrons released from the photocathode, the number of photoelectrons in any general signal can be determined by division of the single photoelectron signal into the larger signal.

1.5 Cosmic Rays

Cosmic rays are charged particles and nuclei originating from astrophysical phenomena. Except for those produced in solar flares, cosmic rays originate from outside the solar system [12]. There are two classifications of cosmic rays:

1. Primaries: Particles produced and accelerated from astrophysical sources (e.g. electrons, protons, helium, etc.)

2. Secondaries: Particles produced from the interaction of primaries with an interstellar gas molecule, or a molecule in a planets atmosphere (e.g. muons, neutrons, etc.)

1.5.1 Cosmic Ray Muons as Charged Particle Source

Cosmic ray muons are the most numerous charged particles found at sea level [12]. They are produced when a primary (typically a proton) interacts with a molecule in the Earth's atmosphere, causing a hadronic shower consisting of mostly pions [13] [14] (see Figure 1.7). The hadronic cascade is dominated by the production of pions [13]. Neutral pions decay into a pair of gamma rays ($\pi^0 \rightarrow \gamma + \gamma$) that go on to cause electromagnetic showers ($\gamma \rightarrow \mathrm{e}^+ + \mathrm{e}^-$); the products of these electromagnetic showers do not penetrate far into the Earth's atmosphere. Charged pions decay into cosmic ray muons ($\pi^+ \rightarrow \mu^+ + \nu_\mu$ and $\pi^- \rightarrow \mu^- + \bar{\nu}_\mu$). The neutrinos have extraordinarily small

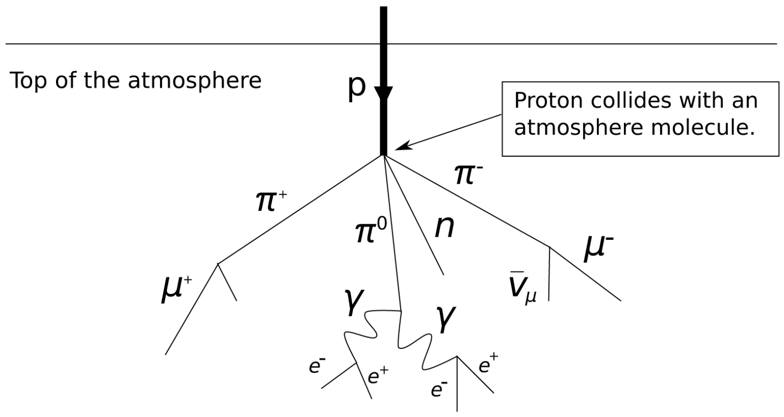


Figure 1.7: Hadronic shower producing cosmic ray muons [9]

interaction cross sections and usually pass through the Earth without interaction; it is the muons that are left in large numbers for observation. On average, one muon hits an area of one square centimetre every minute [13].

Cosmic ray muons reaching the surface of the Earth have an average energy of 4 GeV [12]. At this energy, muons are minimally ionising particles [6]; that is, their mean rate of energy loss as they pass through a material is constant. Because cosmic ray muons are ever-present at the Earth's surface as a reliable and safe source of minimally ionizing particles, they were used to produce the scintillation light in the CDet scintillator bar elements investigated in the work for this thesis.

1.6 Geant4 Simulation Toolkit

GEometry ANd Tracking 4 (Geant4) is a Monte Carlo simulation toolkit, created by CERN, for simulating the passage of radiation through matter. It is a collection of libraries, written in C++, that are used to create powerful simulations of the geometry of a given system, and the resulting interactions of radiation with materials that take place in the system.

1.7 Specific Thesis Goals

The goal of the work presented in this thesis is to measure the number of photoelectrons produced by the photocathode from scintillation light produced from the interaction of cosmic ray muons. Photoelectron dependence on the transverse and longitudinal positioning of the signal within a CDet scintillating bar element were tested. A comparison between the originally proposed $Ti0_2$ coating and the newly proposed 0.25 mm thick aluminized Mylar will be made.

Additionally, an attempt at simulating the results using the Geant4 toolkit is presented in an appendix.

Chapter 2

Experimental Materials and Procedure

The measurement of the number of photoelectrons produced per scintillation event requires the development of appropriate experimental methodology. The pedestal (i.e. noise) must be measured, as well as the single photoelectron signal and the signals produced from cosmic ray muons passing through the scintillator bar.

2.1 Experimental Hardware

A schematic diagram of the data acquisition hardware is presented in Figure 2.1. This diagram displays how each piece of hardware was connected within the overall data acquisition system.

In order to ensure that an isolated region of the bar was responsible for the signal measured in the PMT, two coincidence systems were designed and constructed. In both cases, two scintillating detectors (termed “fingers”) were placed around the scintillating bar, one above and one below (see Figure 2.2 and Figure 2.3). A signal was then produced in the scintillating bar by a cosmic ray muon only if the top and bottom fingers were found to have fired within 30 ns of each other, establishing a coincidence. There were two configurations of fingers. The first configuration used long, thin rectangular fingers with dimensions of 9.5 cm $\times$ 1.2 cm $\times$ 1.2 cm (figure 2.2).

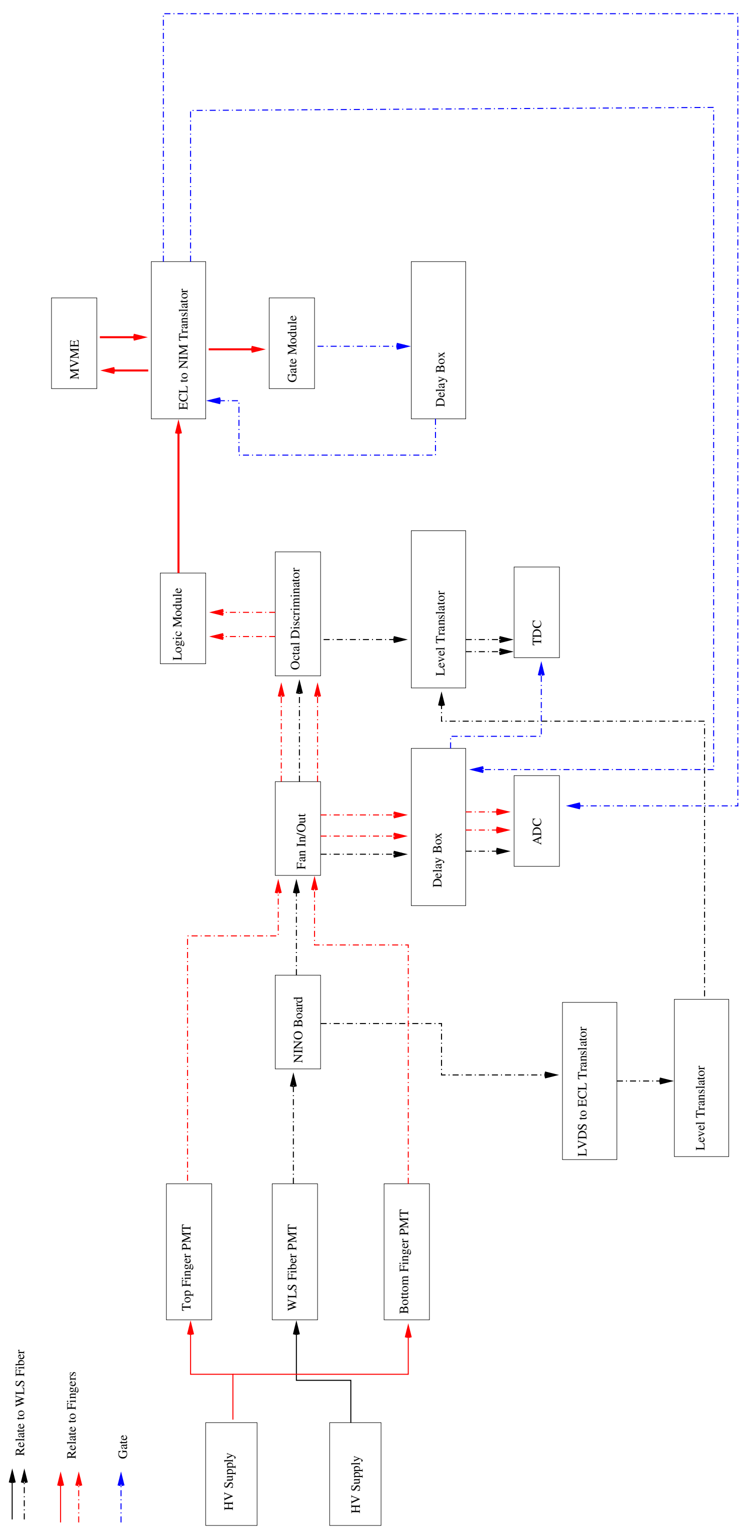


Figure 2.1: Schematic diagram of hardware used to collect data.

Using this configuration of fingers, two sets of data were collected - one with the WLS fibres glued directly to the face of the PMT (which will be referred to as the “glued”

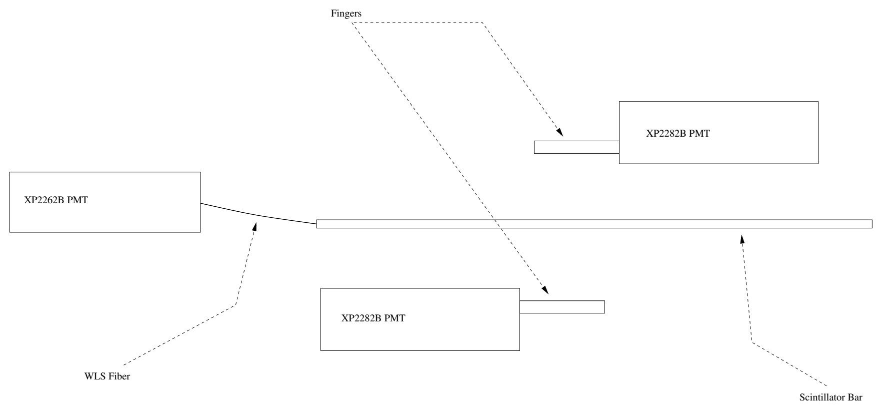


Figure 2.2: Setup used to collect longitudinal and transverse position data.

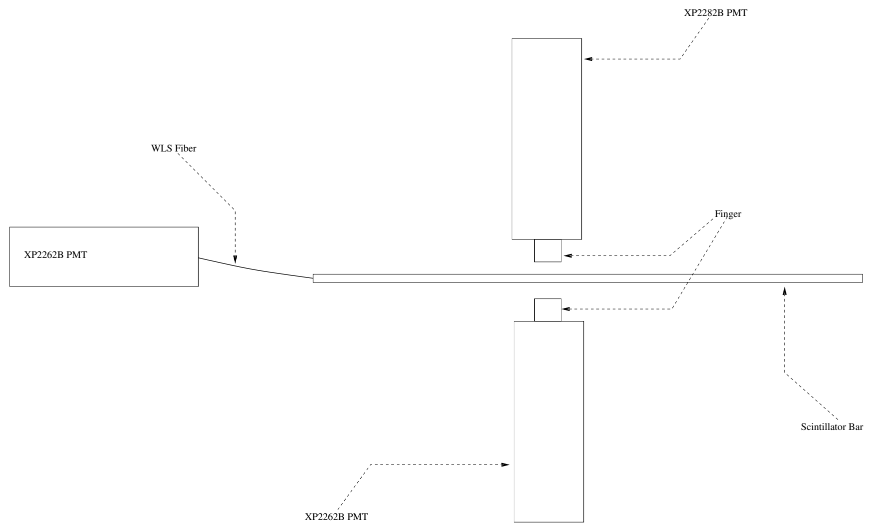


Figure 2.3: Setup used to collect only longitudinal position data.

dataset) and one where the WLS fibres were glued into a plastic disk, which was then taped to the face of the PMT using electrical tape after applying a thin layer of optical grease to the face of the PMT (this will be referred to as the "greased" dataset)[1]. This configuration was used to measure longitudinal and transverse positioning data. The second configuration used cube shaped fingers measuring 25.4 mm × 25.4 mm × 25.4 mm (see Figure 2.3). This configuration was used only to measure longitudinal positioning data, and will be referred to as the "NewSystem" dataset - the fibres were attached to the PMT as in the "greased" data set.

[1]This was done because the fibres kept snapping off the face of the PMT when glued

The bar used in the work for this thesis had two holes extruded along its length, rather than one (as will be used in the CDet)[2] (see Figure 2.4).

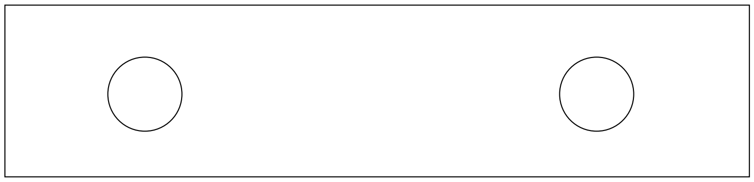


Figure 2.4: Cross section of bar used in experiment.

Additionally, two 1 mm Kururay Y11 WLS fibres were used in place of one 2 mm BCF-92 WLS fiber[3] The effective transmission area is thus half of the area that will be provided by the WLS fibre in the actual CDet configuration; this must be taken into consideration when assessing the final results of this experiment, but the behaviour of photoelectron yields may still be observed. The Y11 WLS fibres used for testing were single clad WLS fibres; these WLS fibres are effective in the same frequency region as the BCF-92 WLS fibres that will be used in the CDet, but it should be noted that they have an emission time four times slower than that of the BCF-92 WLS fibres (12 ns versus 3 ns).

2.1.1 Preparing the Experimental Equipment

The cosmic ray muon coincidence detector used for the work in this thesis needed to be built from scratch. This involved the cutting, cleaning and preparation of all scintillator and optical fibre elements used. Once the materials were properly prepared they were assembled into the completed detector system, ensuring no light leaks were present.

[2]This was a cost saving measure; this bar was already available for testing at JLab. The composition of this scintillator bar is identical to the scintillator bars that will be used in the CDet.

[3]Again, this was a cost saving measure at the time.

Preparing the Scintillating Fingers

Care had to be taken when preparing the fingers to ensure that the surface of the fingers remained clean, smooth and that the edges formed right angles. The following steps were taken in preparing the scintillating finger detectors used for coincidence:

1. A clean work surface was used when working with scintillating plastics. Any potential source of spills or mess was removed, and a clean paper covering was placed on the work surface.

2. The fingers were cut from a sheet of plastic scintillator, using a hand saw. To avoid damaging the material for the fingers, the scintillating plastic was covered with a paper tape and placed between two pieces of wood before being placed in a vice for cutting; care was taken to avoid over-tightening the vice to ensure no small cracks formed.

3. After cutting, the paper tape was removed from the fingers (note: gloves were worn at all stages after this point). The edges of the fingers were then sanded to a smooth finish; the fingers were placed in a felt-lined clamp to hold them in place, and this clamp was secured to the edge of a sturdy workbench. Care was taken to ensure that the fingers were level in the clamp, so that the surfaces of the resulting rectangular prism would be at right angles. The exposed surface of the finger was rinsed with water, and dabbed dry with a clean paper wipe (note: the surface of the plastic scintillator can be damaged if the paper is *wiped*, rather than dabbed). A sheet of 600 grit sandpaper was moistened, and the finger sanded to the maximum smoothness provided by this sandpaper, at

which point the finger was again rinsed with water and pat-dried. This process was repeated for 1200 grit sandpaper, 12 micron sandpaper, 9 micron sandpaper, 3 micron sandpaper and 0.3 micron sandpaper. This entire process was repeated for each side of the finger roughened by the cutting process.

4. After the sanding process was completed, the surfaces of the fingers were washed with a solution comprising 50% water and 50% methanol (pure methanol would damage the surface of the plastic due to it's rapid evaporation). A clean paper wipe was dipped in this alcohol solution, and gently dabbed on the surface of the finger until clean.

5. The surfaces of the fingers were then polished using a 0.05 micron polishing liquid. A small quantity of the polishing liquid was applied to a polishing felt, and a small quantity of water applied on top of that. This was then gently buffed against the fingers until a smooth finish was obtained.

6. The fingers were each glued to the face of a PMT using UV-curing optical glue. Both rectangular fingers were attached to Philips XP2282B PMTs. One cubic finger was attached to a Philips XP2282B PMT, and the second to a Philips XP2262B PMT (the actual PMT used is irrelevant, as these detectors were only used to establish a coincidence, as described in section 2.1).

7. To reduce scintillation light loss, the fingers were covered with a reflective coating. The rectangular fingers were wrapped in one layer of aluminized Mylar; to do this, one end of the finger was first covered in a layer of aluminized Mylar, and the length of the finger was then wrapped in one layer of the Mylar,

using thin pieces of tape to tape the aluminized Mylar to itself. The conductive surface of the aluminized Mylar faced outwards when wrapped, and half a centimetre of plastic scintillator was left exposed near the end destined to be attached to the face of the PMT; this exposed portion of the finger was wrapped with kevlar tape (in order to avoid sparking between the aluminized Mylar and PMT face). The cubic fingers were painted with three coatings of reflective paint; the surface that was to be attached to the face of the PMT was left unpainted.

8. To ensure light tightness of the finger detectors, each finger was wrapped with a layer of Tedlar. The exposed face of the PMT was also covered with Tedlar, and the entire assembly then cleanly and tightly wrapped with a layer of black electrical tape.

9. The assembled finger detectors were then attached to a pico-ammeter, and the high voltage applied to the PMT in increments of no more than -500 V. If a current of 1 μA or more was measured on the pico-ammeter, this indicated a light leak, and the process of wrapping the fingers with Tedlar and electrical tape had to be reversed and redone carefully.

Preparing the Scintillator Bars

Two scintillator bars were tested; one with the originally proposed TiO_2 coating, and one that had been machined and wrapped with 0.25 mm thick aluminized Mylar. Both of these bars were prepared for testing as follows:

1. (*Machined bar only*): The exposed plastic surfaces of the scintillator bar were

washed with a solution of 50% water and 50% methanol, in the same method as described for the fingers. The bar was then wrapped with one layer of aluminized Mylar. Using small pieces of scotch tape, the aluminized Mylar was taped to itself, securing it in position around the bar; care was taken when taping the aluminized Mylar to reduce the number of wrinkles in the material (gloves were worn for this process).

2. The scintillator bars were then wrapped in two layers of Tedlar, to ensure no light leaks would be present. Again, scotch tape was used to smoothly tape the Tedlar to itself. A Tedlar layer was also applied over the end of each bar. Afterwards, one of the two holes in the bars was exposed by cutting a small hole through the Tedlar with a razor (only at one end of the bar, leaving the other side of the hole closed).

Preparing the WLS Fibres

The following steps were taken in preparing the WLS fibres for use (gloves were worn throughout the process):

1. Two 75 cm lengths of WLS fibre were cut from a large spool, using a razor blade.

2. The ends of each WLS fibre were sanded and polished, in a similar process to that described above for the scintillating fingers. 12 micron sandpaper, 9 micron sandpaper, 3 micron sandpaper and 0.3 micron sandpaper were used, as was a 0.05 micron liquid polishing agent. Special care was taken to ensure that the fibres were not excessively bent, so as to avoid cracks forming in the cladding.

A magnifying glass was used to visually inspect for any roughness or burs at the ends of the fibres, as well as for cracks in the cladding; more polishing was performed if any were found.

2.1.2 Assembling the Detector

A metal frame was assembled to hold all the components needed for the detector system. A metal beam was attached vertically to a heavy metal base, and a large metal brace was bolted horizontally to this vertical beam (the scintillating bars would later rest on this horizontal brace as they were being tested). On this horizontal brace, a sliding bracket was placed that allowed for transverse and longitudinal positioning of the scintillating bars. Above and below the large horizontal brace, two smaller horizontal braces were bolted (these would hold the scintillating finger detectors in place). This metal structure was securely clamped to a workbench.

The PMTs of the scintillating finger detectors were placed inside of a metal shield, and the ends wrapped in aluminium foil, to shield them from electrical noise in the lab as well as to provide grounding. They were then placed in their respective brackets, ensuring vertical alignment between the fingers themselves, and strapped securely into place (see Figure 2.2).

In the case of the second configuration of the setup, these same steps were repeated for the scintillating finger detectors, but the horizontal brackets were removed and the scintillating finger detectors strapped to the vertical beam itself (see Figure 2.3).

One end of the WLS fibres were attached to the centre of a Philips XP2262B PMT, using either UV-curing optical glue or optical grease as described in Section 2.1. A blackened metal shield was then fitted with dense black foam at one end, through which a small hole was cut. A black rubber tube was inserted through this hole, and the free ends of the WLS fibres passed through the tube so that the face of the PMT rested against the foam inside the blackened metal shield. The space between the body of PMT and the blackened metal shield was then filled with small pieces of black foam to further protect from a light leak. A 12 cm black rubber sleeve was then slid over the ends of the WLS fibres and inserted partially inside the protruding rubber tube from the metal shield; the junction between the tube and the sleeve was secured with black electrical tape. The free ends of the WLS fibres were inserted into the exposed hole in the scintillator bar, while ensuring that they did not twist around one another in the process. The rubber sleeve around the WLS fibres was partially inserted into the hole in the bar; black foam was then placed surrounding the junction between the sleeve and the bar, and tightly taped into place using black electrical tape to prevent light leaks. Figure 2.5 shows an example of the completed detector assembly. Lights leaks were checked using a pico-ammeter as described in Section 2.1.1.

2.1.3 Hardware System

The electronics used to record the experimental data are shown schematically in Figure 2.1. The PMT attached to the WLS fibres had it's own HV power supply, and the PMTs attached to the fingers shared a separate HV power supply. The electronics path for each set of detectors is described below:

- Fingers: The signals from the top and bottom finger PMTs are sent to a fan

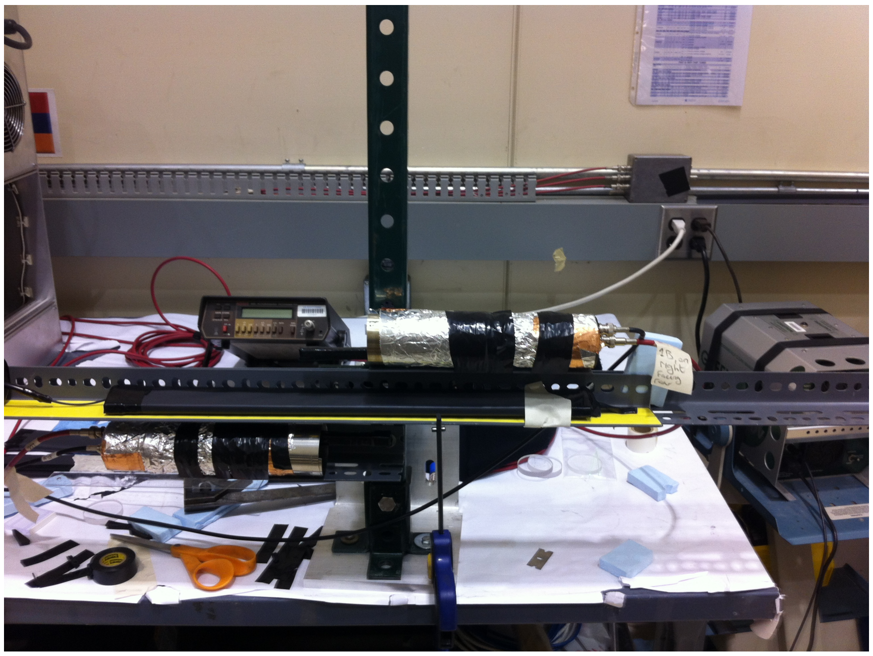


Figure 2.5: Picture of experimental setup.

in/out for duplication. Afterwards, one set of signals from each is sent to a delay box, which is timed to send the signals to an analogue to digital converter (ADC) when a gate signal is received at the ADC. The other set of signals is sent to a discriminator, and if they fall above the threshold of noise they are sent to a logic module to make a digital pulse. This digital pulse is then sent to a ECL/NIM signal translator, and a signal is sent out to a MVME module to signal to the data collection software that it should be ready for readout. Then a signal is sent from the ECL/NIM translator to a gate generator, which produces a gate signal that is delayed through a delay box, translated back to a NIM signal, and sent to provide an integration gate for the ADC and a start signal for the time to digital converter (TDC).

- WLS Fibres: The raw signal from the PMT attached to the WLS fibres was sent to a NINO board for integration[4]. From there, one signal was sent to a fan in/out and another to a LVDS/ECL translator. The signal that was sent to the translator was fed through two more translators, and sent to the TDC for collection. The signal sent to the fan in/out was duplicated, with one signal being delayed and sent to the ADC for collection; the other duplicated signal was sent to a discriminator, which was then translated and sent to the TDC for the final timing measurement.

2.2 Data Acquisition

Data acquisition was handled by a data collection program written at JLab called CODA.

2.3 Experimental Procedure

A standard procedure was implemented to ensure consistent measurements between experimental runs.

2.3.1 Measuring the Pedestal

The pedestal is the effective “noise” in the data acquisition system. The ADC works by charging a capacitor during a specified collection period specified by the supplied gate signal - larger incoming signals produce larger charges on the capacitor, corresponding to larger digital outputs. However, there is always a small trickle of charge

[4]the NINO board provides roughly 10 ns of integration, smoothing the longer emission time of the Y11 WLS fibres.

entering the ADC capacitor during this collection period independent of the input signal resulting from the scintillation light; it is this small current that is responsible for the ADC pedestal. In the absence of an external signal, the only charge on the capacitor will be due to this trickle of background current. To measure the pedestal, all that is required is to set the ADC to collect data, but not actually send any signals to the ADC.

To measure the pedestal, the PMT attached to the WLS fibres remained off while the data acquisition system was set to trigger in the event that either scintillating finger detector saw a signal (this merely expedited the process of collecting the pedestal data). This was left to run for approximately five minutes, after which collection was ceased. The resulting data represented the ADC pedestal.

2.3.2 Measuring Cosmic Ray Muon Signals

Measurement of scintillation produced by passing cosmic ray muons proceeded as follows:

1. The scintillating bar was positioned between the fingers at the desired transverse and longitudinal position. The sliding bracket was clamped securely into place.

2. The high voltage was slowly increased to operational levels for each PMT.

3. The CODA software was run to collect data. It was run for a minimum of five hours to collect sufficient statistics in each run.

4. The bar was repositioned and the above steps were repeated.

When changing between the $Ti0_2$ coated bar and the wrapped bar, the steps outlined in Section 2.1.2 were followed.

2.3.3 Measuring Single Photoelectron Signals

Due to thermal energy of electrons at the photocathode, single photoelectrons are frequently emitted by a process known as thermionic emission [11]. These single photoelectrons compose a significant portion of the noise in the PMT itself; careful detection of the signal produced by the noise can thus give a measure of the single photoelectron value.

To accomplish this careful measurement, a discriminating device known as a NINO board was used. The NINO board is capable of triggering the data acquisition system on very small signals, which makes it the ideal choice for detecting single photoelectron signals in a quick and convenient fashion. The system was set to trigger off of the NINO board, and left to collect data for roughly two minutes to collect sufficient statistics. The resulting ADC spectra consisted of single photoelectron signals[5].

[5]Due to the extremely low flux of cosmic ray muons, their contribution to these measurements was negligible, and their higher energy deposition would ensure that their resulting signals would never be confused with single photoelectron signals

Chapter 3

Analysis

The pedestal data, raw signal data and single photoelectron signal data collected by the data acquisition system provides the requisite information required to analyse the performance of the CDet scintillator bar element. ROOT, an analysis software developed by CERN, facilitated the production of histograms from the data, as well as the fitting of functions to the resulting histograms. These fitted histograms provide a means by which to carry out a performance assessment.

3.1 Reduced χ^2 Fitting

To determine the parameters of a pre-supposed functional form which would best describe any given data set, a statistical method known as reduced χ^2 fitting was used.

The chi-square statistic is defined as follows [15]:

$$\chi^2 = \sum_i \left(\frac{1}{\sigma_i^2} \left[y_i - f(x_i) \right]^2 \right) \tag{3.1}$$

Where $i \in [1, N]$, N is the number of data points, σ_i^2 is the variance of the data point in question, y is the independent (measured) variable, x is the dependent variable, f is the proposed relationship between y and x, y_i is the observed mean and $f(x_i)$ is the predicted mean. In a sense, the chi-square is simply the difference between the experimental and hypothesised values, normalized by error (through the division by variance). On average, χ^2 will be equal to the number of points being modelled - that is, if a data set consists of 30 data points, then χ^2 of an ideally fitted function would also be 30[1] [16].

The reduced χ^2 statistic is defined by normalizing χ^2 by the number of degrees of freedom in the proposed function:

$$\eta = N - p \tag{3.2}$$

Where η is the number of degrees of freedom, and p is the number of adjustable parameters in the proposed model. From this, the reduced χ^2 statistic is defined as:

$$\chi^2_{red} = \frac{\chi^2}{\eta} \tag{3.3}$$

An ideally fitted function will have χ^2_{red} of 1, or below [15][16].

The ROOT analysis software handled the fitting of desired functional forms to the data. Visual inspection of the reported χ^2_{red} value was performed, and parameters of the plot adjusted if the value of χ^2_{red} was significantly larger than 1.

3.2 Determining Value of Pedestal

A Gaussian function was fitted to the pedestal data (example given in Figure 3.1) using the ROOT histogram class methods. After an appropriate χ^2_{red} was found, the

[1]This is assuming the function has not been *overfitted* to the data.

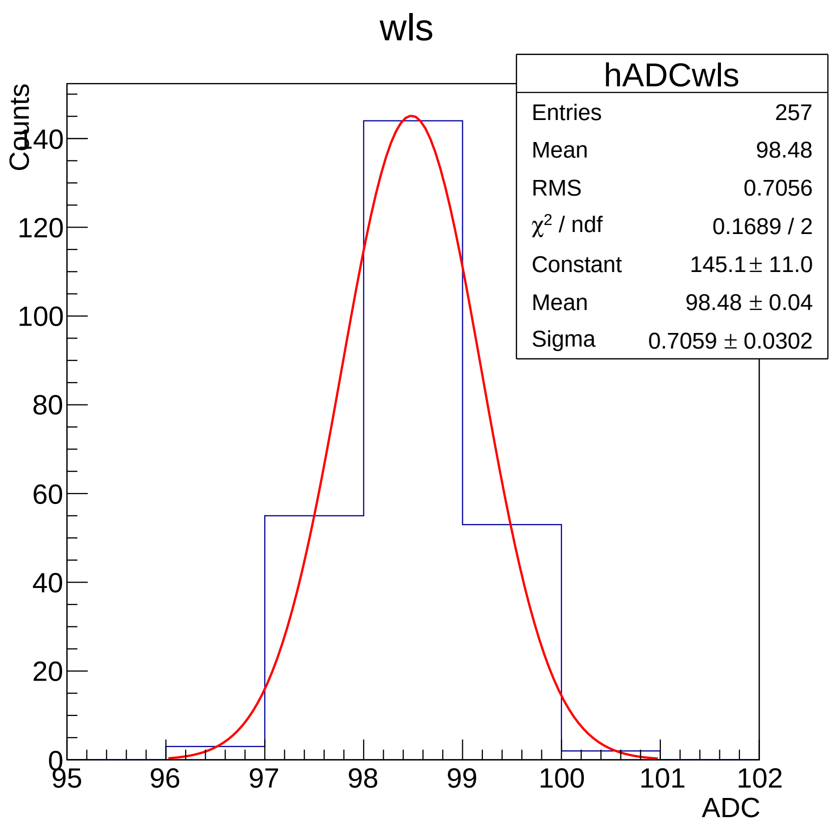


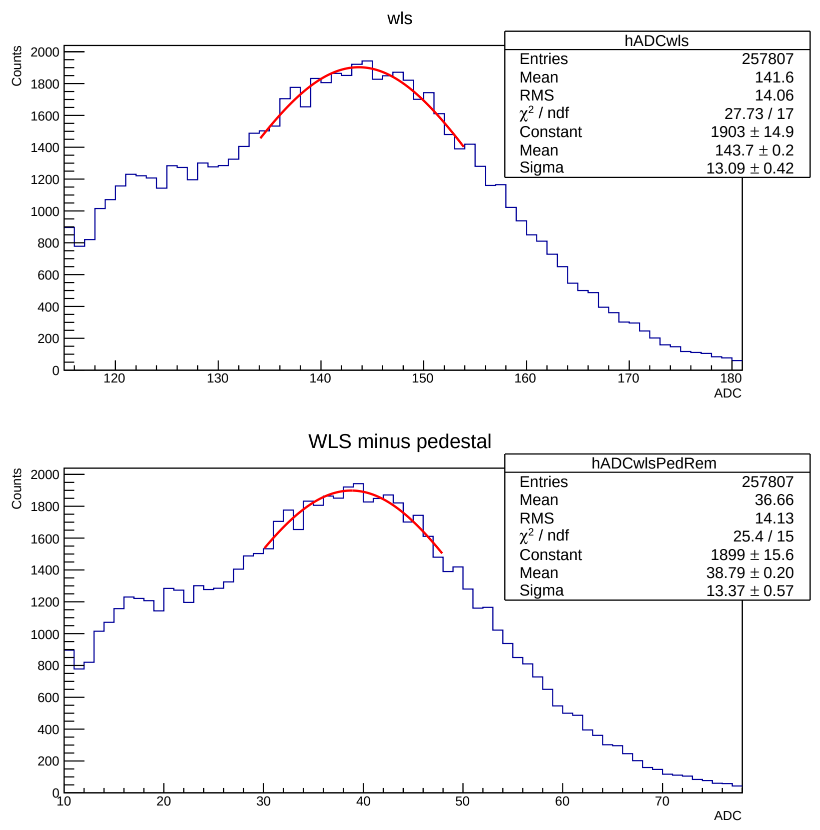
Figure 3.1: Example of Gaussian fitted to pedestal data.

mean of the resulting Gaussian gave a value for the pedestal signal within the ADC.

3.3 Determining Value of Single Photoelectron Signal

The single photoelectron measurements resulted in histograms with a low-energy trailing edge, followed by a peak (see Figure 3.2). The average single photoelectron value is found by fitting a Gaussian to the peak (ignoring the tail), and taking 80% of the Gaussian's mean [17]. This was done after subtracting off the pedestal.

To fit the Gaussian to the peak, the range of interest was manually set so that



hADCwls	
Entries	257807
Mean	141.6
RMS	14.06
χ^2 / ndf	27.73 / 17
Constant	1903 ± 14.9
Mean	143.7 ± 0.2
Sigma	13.09 ± 0.42

hADCwlsPedRem	
Entries	257807
Mean	36.66
RMS	14.13
χ^2 / ndf	25.4 / 15
Constant	1899 ± 15.6
Mean	38.79 ± 0.20
Sigma	13.37 ± 0.57

Figure 3.2: Example of Gaussian fitted to single photoelectron data.

ROOT would only consider the peak when fitting the function, and the reduced χ^2 value was minimized. The mean of the Gaussian, as reported by ROOT, was then used to get the average single photoelectron value as described above.

3.4 Calculating Total Photoelectrons from Cosmic Ray Muon Signals

Due to the small thickness of the scintillating bars being tested, the resulting spectra from the cosmic ray muons form a Landau distribution [6] (see Figure 3.3). As described above for the Gaussian fits, the reduced χ^2 value was minimized while

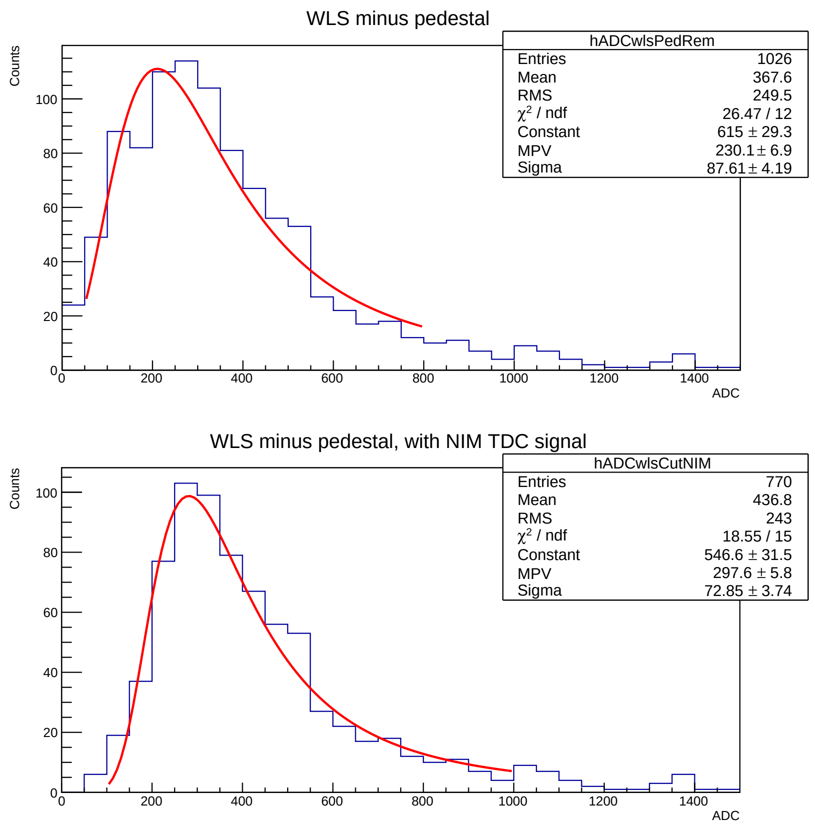


Figure 3.3: Example of Landau distribution fitted to cosmic ray data.

fitting a Landau distribution fitted to each set of cosmic ray muon data. The most probable value of the Landau distribution was then taken as the most probable cosmic ray signal (after subtracting off the pedestal), and this was used in the proceeding analysis.

To determine the average number of photoelectrons responsible for the observed cosmic ray signal, all that needed to be done was to divide the average single photoelectron value into the most probable value of the cosmic ray signal (after subtracting the pedestal value). Since the single photoelectron signal value being used is an average, it's appropriate to assume that all of the photoelectrons produced in the PMT from the cosmic ray signal would individually have this average signal value, and thus the entire signal can be viewed as a superposition of the individual smaller signals.

Additionally, since the cosmic ray muons are minimally ionising, they deposit energy at a constant rate in the plastic scintillator. This allows for the photoelectron yield to be turned into a photoelectron per centimetre of scintillator density, by dividing the photoelectron yield by the thickness of the scintillator bar cross section (this gives units of "# photoelectrons per cm"). This will allow for the photoelectron yield to be considered if the thickness of the scintillating bar elements of the CDet is modified slightly[2].

[2]as mentioned in section 1.1, the scintillator bar elements were changed from a width of 3 cm to 4 cm. This was due to the measured photoelectron yield densities

Chapter 4

Results

4.1 Effect of TiO_2 Coating

The effect of the TiO_2 coating may be seen in Table 4.1.

Table 4.1: Effect of TiO_2 coating.

	"Glued" data	"Greased" data
Yield with machining/wrapping, [pe/cm]	21.3 ± 1.0	12.0 ± 0.9
Yield with TiO_2 coating, [pe/cm]	20.7 ± 1.1	13.4 ± 0.3
Percent difference TiO_2 from machining/wrapping, [%]	-2.8 ± 7.0	11.7 ± 9.0

4.2 Longitudinal Dependence on Photoelectron Yield

Measurements of light yield versus signal distance from PMT may be seen in Figure 4.1. The measurements with the "glued" data set overlapped the empty hole in the bar, whereas the measurements in the "greased" data set did not[1].

The errors on the fit parameters[2] are contained in table 4.2. These measurements were made using the machined/wrapped bar.

[1]The original plan was to obtain more data, and allow direct comparison of yields over hole and yields over no-hole, but due to time constraints this was not possible

[2]"Greased" data set contains insufficient statistics for error estimate on fit parameters.

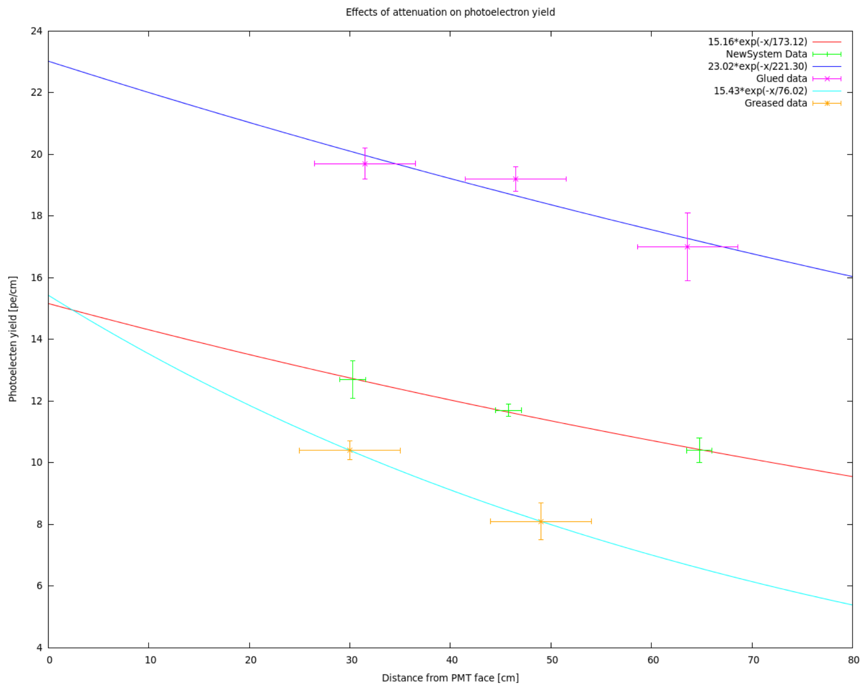


Figure 4.1: Longitudinal measurements of light yield.

Table 4.2: Fit parameters for longitudinal dependence.

Fitted function: $ae^{\frac{-x}{b}}$	"Glued" data	"Greased" data	"NewSystem" data
Initial signal (a), [pe/cm]	23.0 ± 1.7	15.4± N/A	15.2 ± 0.2
Effective attenuation length (b), [cm]	221.3 ± 77.3	76.0± N/A	173.1 ± 7.9

4.3 Transverse Dependence on Photoelectron Yield

Measurements of the transverse dependence on photoelectron yield may be seen in Figure 4.2.

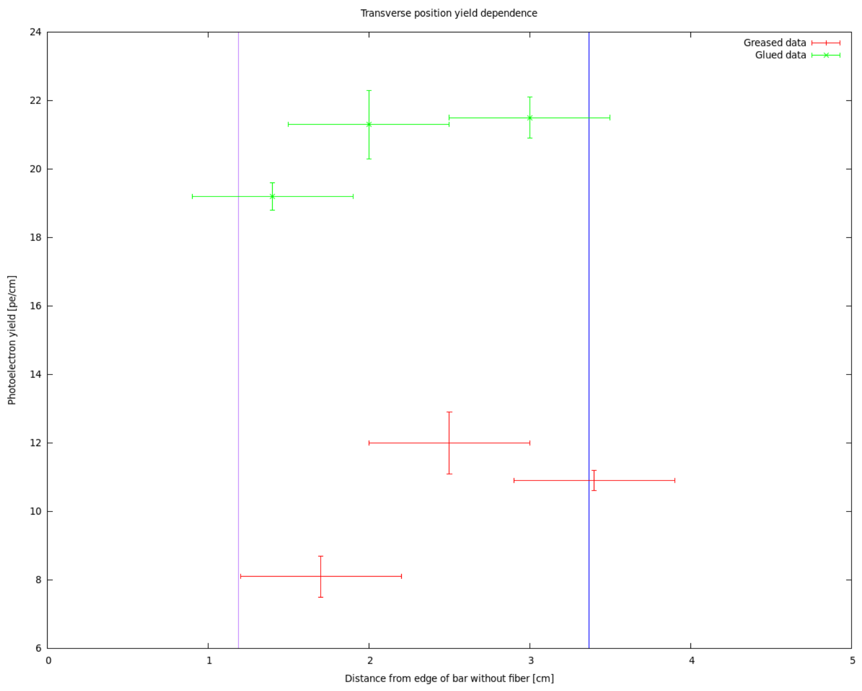


Figure 4.2: Transverse measurements of light yield. The purple vertical line represents finger overlap with the empty hole, while the blue vertical line represents overlap with the hole containing the WLS fibers.

Chapter 5

Conclusions

Referring to Table 4.1, the measured photoelectron yield from the scintillator bar with the TiO_2 coating was approximately the same as that of the scintillator bar that was machined and then wrapped with Mylar. Machining the bars after extrusion and wrapping them with aluminized Mylar will thus not significantly affect the photoelectron yields, providing for efficient performance of CDet elements while allowing the physical construction of the CDet as described in Section 1.1.

Longitudinal dependence of photoelectron yield can be attributed to light attenuation in the WLS fibre (see Figure 4.1). The measured photoelectron yields (which are directly proportional to the signal received at the PMT) follow an effective attenuation curve [18], and the effective attenuation lengths of the "glued" and "NewSystem" datasets agree to within experimental uncertainty (see Table 4.2). This provides support for the conclusion that attenuation inside the WLS fibre is the dominant cause of photoelectron yield degradation longitudinally.

The "glued" and "greased" data sets share the same trend for transverse dependence (see Figure 4.2). Maximum photoelectron yield occurs at a location in the bar in between the empty hole and the hole containing the WLS fibres. There is a slight drop in photoelectron yield for signals overlapping/close to the hole containing the WLS fibres, and a larger drop in yield for signals overlapping/close to the empty hole. This is readily explained as follows: cosmic rays traversing through the portion of the bar between the holes pass through more scintillating material, thereby releasing more energy into the bar and producing more scintillation photons, and so there are more photons available to be captured by the WLS fibres, producing a larger photoelectron yield. Cosmic rays passing through the hole without the WLS fibres pass through less scintillating material, releasing fewer scintillation photons - the same is true for cosmic rays passing through the hole containing the WLS fibres. However, photons are emitted isotropically from the excitation point in the scintillating material, and so the flux of scintillation photons incident on the WLS fibres is inversely proportional to the square of the distance from their point of emission. This means that emissions produced at locations in the scintillating bar closer to the WLS fibres should produce higher photoelectron yields than emissions produced further away from the WLS fibres – exactly what was observed in Figure 4.2.

The difference in observed photoelectron yields from the “glued” versus the “greased” and “NewSystem” data sets (which can be seen in Figures 4.1 and 4.2) is possibly explained by poor technique in attaching the plastic disk to the face of the PMT in the “greased” and “NewSystem” data sets. Since the disk was taped to the face of the PMT with electrical tape, it is possible that the tape stretched over time, allowing the disk to move slightly and for air bubbles to form in the optical grease, which would effect signal transmission to the PMT. Proper methods of attaching the disk containing the WLS fibres to the face of the PMT may have shown less of a disagreement between measured photoelectron yields.

Bibliography

[1] Campbell, Jessica N.A. ”Performance Assessment Tests of Multi-anode Photomultiplier Tube at Jefferson Lab”. Saint Mary’s University: Department of Astronomy and Physics. 2013.

[2] J.J. LeRose, B. Wojtsekhowski, et al. “The Super-Bigbite Spectrometer for Jefferson Lab Hall A.” January 25, 2009. https://userweb.jlab.org/∼bogdanw/SBS-CDR/SBS-CDR.pdf

[3] L. Pentchev, B. Wojtsekhowski. “The Concept and Main Characteristics of the PMT-based Coordinate Detector.” January 11, 2014. https://cnidlamp.jlab.org/SBS-experiments/JDocDB/system/files/biblio/2014/08/cdet_v3.pdf

[4] Sawatzky B., Stepanyan S., et al. “Super BigBite Coordinate Detector (CDet) Review.” February 25, 2014. https://cnidlamp.jlab.org/SBS-experiments/JDocDB/system/files/biblio/2014/08/cdet_review_report-final-v1.pdf

[5] Jones M., Khandaker M., et. al. “ Response to the Coordinate Detector Review Report.” March 22, 2014. https://cnidlamp.jlab.org/SBS-experiments/JDocDB/system/files/biblio/2014/08/responsetocdetreviewreport_mk.pdf

[6] Leo, William R. "Techniques for Nuclear and Particle Physics Experiments: A How-to Approach." Second revised edition. India: Springer. 1994.

[7] Saint-Gobain Crystals. "Premium Plastic Scintillators Data Sheet". http://www.crystals.saint-gobain.com/uploadedFiles/SG-Crystals/Documents/SGC%20BC400-404-408-412-416%20Data%20Sheet.pdf

[8] Saint-Gobain Crystals. "Scintillating Optical Fibres". http://www.crystals.saint-gobain.com/uploadedFiles/SG-Crystals/Documents/SGC%20Fibers%20Brochure.pdf

[9] New World Encyclopedia. "Cosmic Rays". http://www.newworldencyclopedia.org/entry/Cosmic_ray June 24, 2013.

[10] Kurarau. "Scintillation Materials". http://www.phenix.bnl.gov/WWW/publish /donlynch/RXNP/Safety%20Review%206_22_06/Kuraray-PSF-Y11.pdf

[11] Photonis. "Photomultiplier Tube Catalogue". http://lmu.web.psi.ch/docu/manuals/device_manuals/PhotoMultiplier/Photonis.pdf

[12] K.A. Olive et al. "Chinese Physics C" (Particle Data Group). Beijing. Volume 38, No. 9. 2014.

[13] University of Florida Department of Physics. "Cosmic Ray Muons and Muon Lifetime." http://www.phys.ufl.edu/courses/phy4803L/group_I/muon/muon.pdf

[14] Harpell E., et. al. "The CCRT: an inexpensive cosmic ray muon detector." (SLAC). http://www.slac.stanford.edu/cgi-wrap/getdoc/slac-tn-95-001.pdf

[15] Laub and Kuhl. "How Bad is Good? A Critical Look at the Fitting of Reflectivity Models using the Reduced Chi-Square Statistic". http://neutrons.ornl.gov/workshops/sns_hfir_users/posters/Laub_Chi-Square_Data_Fitting.pdf

[16] http://www.physics.csbsju.edu/stats/chi_fit.html

[17] Shepherd M., A. Pope. "Investigation of BURLE 8854 Photomultiplier Tube". August 26, 1997.

[18] M. David, A. Gomes et. al. "Comparative Measurements of WLS Fibers". https://userweb.jlab.org/m̃ahbub/HallA/SBS/CDet/WLS/References/MDavid_CERN_ATLAS_WLS_Attenuation_2000.pdf

[19] Geant4 Collaboration. "Geant4 User's Guide for Application Developers." December 5, 2014. http://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/ForApplicationDeveloper/html/

[20] M. Atac, D. Chrisman and J. Park. "Photon Yields in 3HF Scintillating Fibers as a Function of Doping Concentration with visible Light Photon Counters." IEEE Nuclear Science Symposium and Medical Imaging Conference, Volume 4, pages 527-529. (1994).

[21] D.M. Kaplan and D. Amladi. “Toward Minimum Ionizing particle Detection using Scintillating Fibers and Avalanche Photodiodes.” Northern Illinois University. IEEE Transactions of Nuclear Science, Volume 37, Number 3, pages 1100-1101. June 1990.

[22] V. Agoritsas, et al. “Fast Readout of Scintillating Fiber Arrays Using Position-Sensitive Photomultipliers.” Faros Collaboration. CERN-PPE/94-126. 19-07-1994.

Appendix A

Geant4 Simulation

A simulation was attempted using the Geant4 simulation tool kit. Unfortunately, difficulties in using the tool kit were encountered, and the simulation was not a success. However, the principles of Geant4[1] and the goals of the simulation are outlined below.

A.1 Principle "Ingredients" in a Geant4 Simulation

There are three essential classes in a Geant4 simulation:

1. Detector Geometry: This describes the materials and layout of the system that the simulation will describe.
2. Physics List: This includes all the physical processes to be considered by the simulation (e.g. electromagnetic processes, secondary reactions, and so on).
3. Primary Action Generator: This creates the particles that will be incident upon the system; it effectively starts the simulation by "shooting" particles at the geometry.

Geant4 provides base classes that the user can call to include some default properties; however, the user must still declare the three classes described above at the very

[1]Unfortunately, there are precious few resources for Geant4; visit http://geant4.cern.ch or see [19].

minimum. Additional classes may be defined for more in depth simulation results.

A.2 Attempt at Simulation

An attempt was made to simulate the work described in this thesis, with the goal of better understanding how the scintillation photons enter the WLS fibre and propagate to the PMT. Unfortunately, there were difficulties encountered in trying to make the WLS fibre function in the simulation - photons would not enter and propagate down the length of the WLS fibre. An expert at SLAC, Dr. Michael Kelsey, was contacted regarding this issue but no solution was found.

A.3 Geant4 Code

The Geant4 code written for the simulation may be found below; it comprises several files. Each portion of code consists of a header file and the main body of code.

A.3.1 Detector Geometry

Header file

```
/*
 * Written by: Nathan Murtha
 * December 18, 2014
 *
 * Contains the basics needed for detector element construction.
 *
 */

#ifndef ScintDetectorConstruction_h
#define ScintDetectorConstruction_h 1

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

class G4PhysicalVolume; // need to place physical volumes for detector

class ScintDetectorConstruction : public G4VUserDetectorConstruction
```

```
{
    public: // public methods for detector construction
       ScintDetectorConstruction(); //constructor
       virtual ~ScintDetectorConstruction(); //destructor

    public:
       virtual G4VPhysicalVolume* Construct();
};

#endif
```

./Geant4Code/include/ScintDetectorConstruction.hh

Code body

```
/*
 * Written by Nathan Murtha
 * December 18, 2014
 *
 * Specifies the geometry needed for "ScintillatorTest"
 *
 */

#include "ScintDetectorConstruction.hh"
//#include "ScintSteppingAction.hh"
#include "ScintSensitiveDetector.hh"

#include "G4LogicalBorderSurface.hh"
#include "G4LogicalSkinSurface.hh"
#include "G4OpticalSurface.hh"
#include "G4OpBoundaryProcess.hh"

#include "G4Material.hh"
#include "G4Element.hh"
#include "G4NistManager.hh"

#include "G4RunManager.hh"
#include "G4SDManager.hh"

#include "G4Box.hh"
#include "G4Tubs.hh"
#include "G4LogicalVolume.hh"
#include "G4PVPlacement.hh"
#include <G4SubtractionSolid.hh>

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

#include "G4ThreeVector.hh"

#include "G4VisAttributes.hh"

// ==== CONSTRUCTOR ====
ScintDetectorConstruction::ScintDetectorConstruction()
  : G4VUserDetectorConstruction() // just inherit from Geant4 class
{
}

// ==== DESTRUCTOR ====
ScintDetectorConstruction::~ScintDetectorConstruction()
{
}

// ==== CONSTRUCT ALL GEOMETRY ====
G4VPhysicalVolume* ScintDetectorConstruction::Construct()
{
    // ============================================================================== //
    // -------------- Get Required Materials ----------------------- //
    // ============================================================================== //

    // Get NIST materials manager for premade materials
    G4NistManager* NISTmanager = G4NistManager::Instance();

    // Call up premade materials from NIST manager
    G4Material* WorldMaterial = NISTmanager->FindOrBuildMaterial("G4_AIR
    ");

    G4Material* BarMaterial = NISTmanager->FindOrBuildMaterial("
    G4_PLASTIC_SC_VINYLTOLUENE");

    G4Material* FiberCoreMaterial = NISTmanager->FindOrBuildMaterial("
    G4_POLYSTYRENE");

    G4Material* DetectorMaterial = NISTmanager->FindOrBuildMaterial("
    G4_Al");

    // Define materials needed from scratch (PMMA for fiber cladding)
    G4double a, z, density;
    G4int ncomponents, natoms;
    G4String name, symbol;

    a = 12.011*g/mole;
    G4Element* elC = new G4Element(name="Carbon", symbol="C", z=6.0, a);

    a = 1.008*g/mole;
    G4Element* elH = new G4Element(name="Hydrogen", symbol="H", z=1.0, a
    );

    a= 15.999*g/mole;
    G4Element* elO = new G4Element(name="Oxygen", symbol="O", z=8.0, a);

    density = 1.19*g/cm3;
    G4Material* PMMA = new G4Material(name="PMMA", density, ncomponents
    =3);
    PMMA->AddElement(elC, natoms=5);
    PMMA->AddElement(elH, natoms=8);
    PMMA->AddElement(elO, natoms=2);

    // Define Birks Constant for Scintillator
    BarMaterial->GetIonisation()->SetBirksConstant(0.126*mm/MeV);


    // ================================================================== //
    // ------------ GENERATE MATERIALS PROPERTIES TABLES ------------ //
    // ================================================================== //

    const G4int nEntries = 33;

    // ******** Photon energies to interpolate between *******
    G4double PhotonEnergy[nEntries] =
              { 2.066*eV, 2.109*eV, 2.152*eV, 2.195*eV,
              2.238*eV, 2.281*eV, 2.324*eV, 2.367*eV,
              2.410*eV, 2.453*eV, 2.496*eV, 2.539*eV,
              2.582*eV, 2.625*eV, 2.668*eV, 2.711*eV,
              2.754*eV, 2.797*eV, 2.840*eV, 2.883*eV,
              2.926*eV, 2.969*eV, 3.012*eV, 3.055*eV,
              3.098*eV, 3.141*eV, 3.184*eV, 3.227*eV,
              3.270*eV, 3.313*eV, 3.356*eV, 3.399*eV,
              3.444*eV };


    // ********  Air for the world material  *********
    G4double RefractiveIndexWorld[nEntries] =
               { 1.00, 1.00, 1.00, 1.00,
               1.00, 1.00, 1.00, 1.00,
               1.00, 1.00, 1.00, 1.00,
               1.00, 1.00, 1.00, 1.00,
               1.00, 1.00, 1.00, 1.00,
               1.00, 1.00, 1.00, 1.00,
               1.00, 1.00, 1.00, 1.00,
               1.00, 1.00, 1.00, 1.00,
               1.00 };

    G4MaterialPropertiesTable* MPT_World = new G4MaterialPropertiesTable
   ();

    MPT_World->AddProperty("RINDEX", PhotonEnergy, RefractiveIndexWorld,
    nEntries)
             ->SetSpline(true);

    WorldMaterial->SetMaterialPropertiesTable(MPT_World);


    // ********** Scintillator for the bar ***********
    // http://www.crystals.saint-gobain.com/uploadedFiles/SG-Crystals/
   Documents/SGC%20BC400-404-408-412-416%20Data%20Sheet.pdf
    G4double RefractiveIndexBar[nEntries] =
              { 1.58, 1.58, 1.58, 1.58,
              1.58, 1.58, 1.58, 1.58,
              1.58, 1.58, 1.58, 1.58,
              1.58, 1.58, 1.58, 1.58,
              1.58, 1.58, 1.58, 1.58,
              1.58, 1.58, 1.58, 1.58,
              1.58, 1.58, 1.58, 1.58,
              1.58, 1.58, 1.58, 1.58,
              1.58 };
```

```

  G4double AbsorptionLengthBar[nEntries] =
            { 2.1*m, 2.1*m, 2.1*m, 2.1*m,
            2.1*m, 2.1*m, 2.1*m, 2.1*m,
            2.1*m, 2.1*m, 2.1*m, 2.1*m,
            2.1*m, 2.1*m, 2.1*m, 2.1*m,
            2.1*m, 2.1*m, 2.1*m, 2.1*m,
            2.1*m, 2.1*m, 2.1*m, 2.1*m,
            2.1*m, 2.1*m, 2.1*m, 2.1*m,
            2.1*m, 2.1*m, 2.1*m, 2.1*m,
            2.1*m };

  G4double ScintFastComponent[nEntries] =
            { 0.0, 0.0, 0.0, 0.0,
            0.0, 0.0, 0.0, 1.0,
            5.0, 8.0, 10.0, 12.0,
            18.0, 26.0, 34.0, 50.0,
            55.0, 72.0, 94.0, 100.0,
            91.0, 78.0, 50.0, 31.0,
            20.0, 17.0, 10.0, 8.0,
            7.0, 6.0, 6.0, 4.0,
            2.0 };


  G4double ScintSlowComponent[nEntries] =
            { 0.0, 0.0, 0.0, 0.0,
            0.0, 0.0, 0.0, 1.0,
            5.0, 8.0, 10.0, 12.0,
            18.0, 26.0, 34.0, 50.0,
            55.0, 72.0, 94.0, 100.0,
            91.0, 78.0, 50.0, 31.0,
            20.0, 17.0, 10.0, 8.0,
            7.0, 6.0, 6.0, 4.0,
            2.0 };


  G4MaterialPropertiesTable* MPT_Bar = new G4MaterialPropertiesTable()
 ;

  MPT_Bar->AddProperty("RINDEX",          PhotonEnergy,
 RefractiveIndexBar,   nEntries)
            ->SetSpline(true);
  MPT_Bar->AddProperty("ABSLENGTH",       PhotonEnergy,
 AbsorptionLengthBar,  nEntries)
            ->SetSpline(true);
  MPT_Bar->AddProperty("FASTCOMPONENT",PhotonEnergy,
 ScintFastComponent,   nEntries)
            ->SetSpline(true);
  MPT_Bar->AddProperty("SLOWCOMPONENT",PhotonEnergy,
 ScintSlowComponent,   nEntries)
            ->SetSpline(true);

  MPT_Bar->AddConstProperty("SCINTILLATIONYIELD",10000./MeV); // 100eV
 /photon
  MPT_Bar->AddConstProperty("RESOLUTIONSCALE",1.0);
```

```
    MPT_Bar->AddConstProperty("FASTTIMECONSTANT", 1.0*ns);
    MPT_Bar->AddConstProperty("SLOWTIMECONSTANT",10.*ns);
    MPT_Bar->AddConstProperty("YIELDRATIO",0.8); // relative strength of
     fast component as fraction of total scintillation yield

    BarMaterial->SetMaterialPropertiesTable(MPT_Bar);


    // ********** WLS Fiber Core Properties ***********
    // http://www.phenix.bnl.gov/WWW/publish/donlynch/RXNP/Safety%20
   Review%206_22_06/Kuraray-PSF-Y11.pdf
    // http://kuraraypsf.jp/psf/ws.html

    G4double RefractiveIndexFiberCore[nEntries] =
              { 1.59, 1.59, 1.59, 1.59,
              1.59, 1.59, 1.59, 1.59,
              1.59, 1.59, 1.59, 1.59,
              1.59, 1.59, 1.59, 1.59,
              1.59, 1.59, 1.59, 1.59,
              1.59, 1.59, 1.59, 1.59,
              1.59, 1.59, 1.59, 1.59,
              1.59, 1.59, 1.59, 1.59,
              1.59 };

    G4double AbsorptionLengthFiberCore[nEntries] =
              { 3.5*m, 3.5*m, 3.5*m, 3.5*m,
              3.5*m, 3.5*m, 3.5*m, 3.5*m,
              3.5*m, 3.5*m, 3.5*m, 3.5*m,
              3.5*m, 3.5*m, 3.5*m, 3.5*m,
              3.5*m, 0.5*mm, 0.5*mm, 0.5*mm,
              0.5*mm, 0.5*mm, 0.5*mm, 0.5*mm,
              0.5*mm, 0.5*mm, 0.5*mm, 0.5*mm,
              0.5*mm, 0.5*mm, 0.5*mm, 0.5*mm,
              0.5*mm };

    G4double EmissionFiberCore[nEntries] = // relative emission spectrum
              { 2.0, 7.0, 10.0, 15.0,
              20.0, 30.0, 40.0, 50.0,
              80.0, 77.0, 75.0, 75.0,
              98.0, 100.0, 37.0, 8.0,
              1.0, 0.0, 0.0, 0.0,
              0.0, 0.0, 0.0, 0.0,
              0.0, 0.0, 0.0, 0.0,
              0.0, 0.0, 0.0, 0.0,
              0.0};

    G4double ScintillateFiberCore[nEntries] = // relative scintillation
   spectrum
              { 2.0, 7.0, 10.0, 15.0,
              20.0, 30.0, 40.0, 50.0,
              80.0, 77.0, 75.0, 75.0,
              98.0, 100.0, 37.0, 8.0,
              1.0, 0.0, 0.0, 0.0,
              0.0, 0.0, 0.0, 0.0,
              0.0, 0.0, 0.0, 0.0,
                0.0, 0.0, 0.0, 0.0,
                0.0};

    G4MaterialPropertiesTable* MPT_FiberCore = new
    G4MaterialPropertiesTable();

    MPT_FiberCore->AddProperty("RINDEX",        PhotonEnergy,
    RefractiveIndexFiberCore,   nEntries)
                ->SetSpline(true);
    MPT_FiberCore->AddProperty("WLSABSLENGTH", PhotonEnergy,
    AbsorptionLengthFiberCore, nEntries)
                ->SetSpline(true);
    MPT_FiberCore->AddProperty("WLSCOMPONENT", PhotonEnergy,
    EmissionFiberCore,          nEntries)
                ->SetSpline(true);
    MPT_FiberCore->AddProperty("FASTCOMPONENT",PhotonEnergy,
    ScintillateFiberCore,  nEntries) //
                ->SetSpline(true);

    MPT_FiberCore->AddConstProperty("WLSTIMECONSTANT", 0.5*ns);
    MPT_FiberCore->AddConstProperty("SCINTILLATIONYIELD",10000./MeV); //
     100eV/photon
    MPT_FiberCore->AddConstProperty("RESOLUTIONSCALE",1.0);//
    MPT_FiberCore->AddConstProperty("FASTTIMECONSTANT", 1.0*ns);//

    FiberCoreMaterial->SetMaterialPropertiesTable(MPT_FiberCore);

    // ********* WLS Fiber Cladding Properties **********

    G4double RefractiveIndexFiberCladding[nEntries] =
                  { 1.49, 1.49, 1.49, 1.49,
                  1.49, 1.49, 1.49, 1.49,
                  1.49, 1.49, 1.49, 1.49,
                  1.49, 1.49, 1.49, 1.49,
                  1.49, 1.49, 1.49, 1.49,
                  1.49, 1.49, 1.49, 1.49,
                  1.49, 1.49, 1.49, 1.49,
                  1.49, 1.49, 1.49, 1.49,
                  1.49 };

   G4double AbsorptionLengthFiberCladding[nEntries] =
                { 3.5*m, 3.5*m, 3.5*m, 3.5*m,
                3.5*m, 3.5*m, 3.5*m, 3.5*m,
                3.5*m, 3.5*m, 3.5*m, 3.5*m,
                3.5*m, 3.5*m, 3.5*m, 3.5*m,
                3.5*m, 3.5*m, 3.5*m, 3.5*m,
                3.5*m, 3.5*m, 3.5*m, 3.5*m,
                3.5*m, 3.5*m, 3.5*m, 3.5*m,
                3.5*m, 3.5*m, 3.5*m, 3.5*m,
                3.5*m };


   G4MaterialPropertiesTable* MPT_FiberCladding = new
    G4MaterialPropertiesTable();

```

```
  MPT_FiberCladding->AddProperty("RINDEX"    ,PhotonEnergy,
   RefractiveIndexFiberCladding,  nEntries)
                        ->SetSpline(true);
  MPT_FiberCladding->AddProperty("ABSLENGTH",PhotonEnergy,
   AbsorptionLengthFiberCladding, nEntries)
                        ->SetSpline(true);

  PMMA->SetMaterialPropertiesTable(MPT_FiberCladding);

  // ********* OPTICAL SURFACE ***********
/*
  G4double reflectivity[nEntries] =
              { 0.0389, 0.0389, 0.0389, 0.0389,
               0.0389, 0.0389, 0.0389, 0.0389,
               0.0389, 0.0389, 0.0389, 0.0389,
               0.0389, 0.0389, 0.0389, 0.0389,
               0.0389, 0.0389, 0.0389, 0.0389,
               0.0389, 0.0389, 0.0389, 0.0389,
               0.0389, 0.0389, 0.0389, 0.0389,
               0.0389, 0.0389, 0.0389, 0.0389,
               0.0389 };
*/
   // ********* For the detector **********
   G4double RefractiveIndexDetector[nEntries] =
              { 1.59, 1.59, 1.59, 1.59,
               1.59, 1.59, 1.59, 1.59,
               1.59, 1.59, 1.59, 1.59,
               1.59, 1.59, 1.59, 1.59,
               1.59, 1.59, 1.59, 1.59,
               1.59, 1.59, 1.59, 1.59,
               1.59, 1.59, 1.59, 1.59,
               1.59, 1.59, 1.59, 1.59,
               1.59 };

   G4double AbsorptionLengthDetector[nEntries] =
              { 0.01*m, 0.01*m, 0.01*m, 0.01*m,
               0.01*m, 0.01*m, 0.01*m, 0.01*m,
               0.01*m, 0.01*m, 0.01*m, 0.01*m,
               0.01*m, 0.01*m, 0.01*m, 0.01*m,
               0.01*m, 0.01*m, 0.01*m, 0.01*m,
               0.01*m, 0.01*m, 0.01*m, 0.01*m,
               0.01*m, 0.01*m, 0.01*m, 0.01*m,
               0.01*m, 0.01*m, 0.01*m, 0.01*m,
               0.01*m };

   G4MaterialPropertiesTable* MPT_Detector = new
   G4MaterialPropertiesTable();
   MPT_Detector->AddProperty("RINDEX",        PhotonEnergy,
   RefractiveIndexDetector,nEntries)
                  ->SetSpline(true);
   MPT_Detector->AddProperty("ABSLENGTH",     PhotonEnergy,
   AbsorptionLengthDetector,      nEntries)
                  ->SetSpline(true);

   DetectorMaterial->SetMaterialPropertiesTable(MPT_Detector);
```

```

    // ================================================================ //
    // ------------------- BUILD WORLD ------------------------------- //
    // ================================================================ //

    // World dimensions
    G4double WorldLength = 400.0*cm; // z-axis
    G4double WorldWidth = 400.0*cm; // x-axis
    G4double WorldHeight = 400.0*cm; // y-axis

    // Make solid shape
    G4Box* World_Solid = new G4Box("World", // name of solid
                                   0.5*WorldWidth, // half-width, x
                                   0.5*WorldHeight, // half-width, y
                                   0.5*WorldLength);// half-width, z

    // Make logical volume out of world shape
    G4LogicalVolume* World_Logic = new G4LogicalVolume(World_Solid,
                                           WorldMaterial,
                                           "World");

    // Make physical volume of the worlds logical volumes
    G4VPhysicalVolume* World_Physical = new G4PVPlacement(0, // no
   rotation
                                            G4ThreeVector(), // at
   origin
                                            World_Logic, // associate
   world logical volume
                                            "World", // its name
                                            0, // null mother volume
                                            false, // no boolean
   operation
                                            0); // copy number


    // ================================================================ //
    // ------------------- BUILD BAR --------------------------------- //
    // ================================================================ //

    // Bar dimensions
    G4double BarLength = 55.2*cm; // along z-axis
    G4double BarHeight = 0.81*cm; // along y-axis
    G4double BarWidth = 4.51*cm; // along x-axis

    // Make bar with holes cut out
    G4Box* Block_Solid = new G4Box("Block", // name of solid
                                   0.5*BarWidth,  // half-width, x
                                   0.5*BarHeight, // half-width, y
                                   0.5*BarLength);// half-width, z

    G4Tubs* hole = new G4Tubs("hole",
                              0*mm, // radius start
                              1.5*mm, // radius end
                              0.6*BarLength, // length, make longer than
     bar to avoid ambiguity at ends
```

```
                              0*deg, // start at 0 degrees...
                              360*deg); // ... spin through to 360 degrees


 G4SubtractionSolid* Bar_Intermediate =
            new G4SubtractionSolid("Block_Solid-hole",
                                    Block_Solid, // take block...
                                    hole,  // ... and subtract cylinder to make hole
                                    0,
                                    G4ThreeVector(-1.265*cm, 0.0, 0.0)); // location of subtraction

 G4SubtractionSolid* Bar_Solid =
            new G4SubtractionSolid("Bar_Intermediate-hole", // make another hole
                                    Bar_Intermediate,
                                    hole,
                                    0,
                                    G4ThreeVector(1.315*cm, 0.0, 0.0));


 // Make logical volume out of bar shape
 G4LogicalVolume* Bar_Logic = new G4LogicalVolume(Bar_Solid, // with holes in it
                                       BarMaterial,
                                       "Bar");

 // Make the bar grey, so we can see it easier
 G4Colour grey(0.7, 0.7, 0.7);
 G4VisAttributes* BarVisAttributes = new G4VisAttributes(grey);
 Bar_Logic->SetVisAttributes(BarVisAttributes);

 // Place bar
 G4VPhysicalVolume* Bar_Physical = new G4PVPlacement(0, // no rotation
                                        G4ThreeVector(0.0, 0.0, 0.0), // at world origin
                                        Bar_Logic, // associate bar logical volume
                                        "Bar", // its name
                                        World_Logic, // in the world volume
                                        false, // no boolean operation
                                        0); // copy number


 // ============================================================ //
 // ------------------ BUILD WLS FIBER -------------------------- //
 // ============================================================ //

 G4double FiberLength = 75.0*cm; // along z-axis, will protrude from
bar
 G4double FiberCoreInnerRadii = 0.00*mm;
 G4double FiberCoreOuterRadii = 0.94*mm;
 G4double FiberCladdingInnerRadii = 0.00*mm;
 G4double FiberCladdingOuterRadii = 1.00*mm;
 G4double FiberStartAngle = 0.0*deg;
 G4double FiberEndAngle   = 360.0*deg;

 // Solids
 G4Tubs* FiberCladding_Solid = new G4Tubs("FiberCladding",
                             FiberCladdingInnerRadii, // radius start
                             FiberCladdingOuterRadii, // radius end
                             0.5*FiberLength, // length
                             FiberStartAngle, // start at 0 degrees...
                             FiberEndAngle); // ... spin through 360
degrees


 G4Tubs* FiberCore_Solid = new G4Tubs("FiberCore",
                             FiberCoreInnerRadii, // radius start
                             FiberCoreOuterRadii, // radius end
                             0.5*FiberLength, // length
                             FiberStartAngle, // start at 0 degrees...
                             FiberEndAngle); // ... spin through 360
degrees

 // Logical volumes
 G4LogicalVolume* FiberCladding_Logic = new G4LogicalVolume(
FiberCladding_Solid,
                                            PMMA,
                                            "FiberCladding");


 G4LogicalVolume* FiberCore_Logic = new G4LogicalVolume(
FiberCore_Solid,
                                            FiberCoreMaterial,
                                            "FiberCore");

 // Make the fiber core green and cladding purple, so we can see it
easier
 G4Colour green(0.5, 1.0, 0.0);
 G4VisAttributes* FiberCoreVisAttributes = new G4VisAttributes(green)
;
 FiberCore_Logic->SetVisAttributes(FiberCoreVisAttributes);

 G4Colour purple(0.8, 0.0, 0.8);
 G4VisAttributes* FiberCladdingVisAttributes = new G4VisAttributes(
purple);
 FiberCladding_Logic->SetVisAttributes(FiberCladdingVisAttributes);

 // Place fiber
 G4VPhysicalVolume* FiberCladding_Physical = new G4PVPlacement(0, //
no rotation
```

```
                                                   G4ThreeVector(-1.265*cm,
    0.0, 9.9*cm), // in bar hole, moved forward to protrude from only
    one end
                                                   FiberCladding_Logic, //
    associate fiber logical volume
                                                   "FiberCladding", // its name
                                                   World_Logic, // relative to
    world
                                                   false, // no boolean
    operation
                                                   0); // copy number

    G4VPhysicalVolume* FiberCore_Physical = new G4PVPlacement(0, // no
    rotation
                                                   G4ThreeVector(-1.265*cm,
    0.0, 9.9*cm), // in bar hole
                                                   FiberCore_Logic, //
    associate fiber logical volume
                                                   "FiberCore", // its name
                                                   World_Logic, // in the world
     volume
                                                   false, // no boolean
    operation
                                                   0); // copy number


    // ==================================================================== //
    // ---------- Define optical surfaces ------------------------------- //
    // ==================================================================== //
    // http://refractiveindex.info/legacy/?group=PLASTICS&material=PMMA
/*
    G4OpticalSurface* OpSurfaceOut = new G4OpticalSurface("claddingOut")
    ;

    OpSurfaceOut->SetType(dielectric_dielectric);
    OpSurfaceOut->SetModel(unified);
    OpSurfaceOut->SetFinish(polished);

    G4LogicalBorderSurface* claddingOut = new
                    G4LogicalBorderSurface("cladding_out",
                                                   FiberCladding_Physical,
                                                   FiberCore_Physical,
                                                   OpSurfaceOut);

    G4MaterialPropertiesTable* MPT_OpSurfaceOut = new
    G4MaterialPropertiesTable();

    MPT_OpSurfaceOut->AddProperty("RINDEX", PhotonEnergy,
    RefractiveIndexFiberCladding, nEntries);
    //MPT_OpSurfaceOut->AddProperty("REFLECTIVITY", PhotonEnergy,
    reflectivity, nEntries);

    OpSurfaceOut->SetMaterialPropertiesTable(MPT_OpSurfaceOut);
*/
    // ==================================================================== //
    // ---------------------- BUILD DETECTOR ---------------------------- //
    // ================================================================== //

    // Detector dimensions
    G4double DetectorLength = 2.0*mm;

    // Make solid shape
    G4Tubs* Detector_Solid = new G4Tubs("Detector", // name of solid
                                        FiberCoreInnerRadii,
                                        FiberCladdingOuterRadii,
                                        0.5*DetectorLength,
                                        FiberStartAngle,
                                        FiberEndAngle);

/*
    G4Box* Detector_Solid = new G4Box("Detector", // name of solid
                                   0.5*BarWidth,  // half-width, x
                                   0.5*BarHeight, // half-width, y
                                   0.5*DetectorLength);// half-width, z
*/
    // Make logical volume out of bar shape
    G4LogicalVolume* Detector_Logic = new G4LogicalVolume(Detector_Solid
   ,
                                             DetectorMaterial,
                                             "Detector");

    // Place detector
    G4VPhysicalVolume* Detector_Physical = new G4PVPlacement(0, // no
   rotation
                                              G4ThreeVector(-1.265*cm,
   0.0, 47.5*cm), // in world
                                              //G4ThreeVector
   (0.0,0.0,27.7*cm),
                                              Detector_Logic, // associate
    bar logical volume
                                              "Detector", // its name
                                              World_Logic, // in the world
    volume
                                              false, // no boolean
   operation
                                              0); // copy number

    G4Colour brown(0.7, 0.4, 0.1);
    G4VisAttributes* DetectorVisAttributes = new G4VisAttributes(brown);
    Detector_Logic->SetVisAttributes(DetectorVisAttributes);

    // ================================================================== //
    // --------------- IMPLEMENT STEPPING ACTION ------------------------ //
    // ================================================================== //
/*
    ScintSteppingAction* steppingAction = ScintSteppingAction::Instance
   ();
    steppingAction->SetVolume(Bar_Logic);
*/
    // ================================================================== //
    // ------------- MAKE SENSITIVE DETECTORS ----------------------- //
    // ================================================================ //

    G4SDManager* SDmanager = G4SDManager::GetSDMpointer();

    ScintSensitiveDetector* detector = new ScintSensitiveDetector("Detector");
    SDmanager->AddNewDetector(detector);
    Detector_Logic->SetSensitiveDetector(detector); //detector logical volume is sensitive

    // ================================================================ //
    // ------------- RETURN THE PHYSICAL WORLD ---------------------- //
    // ================================================================ //

    return World_Physical;
}
```

./Geant4Code/src/ScintDetectorConstruction.cc

A.3.2 Physics List

Header file

```
//
// Contains physical processes to be used in simulation
//
// $Id$
//
//----------------------------------------------------------------

#ifndef ScintPhysicsList_h
#define ScintPhysicsList_h 1

#include "G4OpWLS.hh"
#include "G4Cerenkov.hh"
#include "G4Scintillation.hh"
#include "G4OpMieHG.hh"
#include "G4OpRayleigh.hh"
#include "G4OpAbsorption.hh"
#include "G4OpBoundaryProcess.hh"

#include "globals.hh"
#include "G4VUserPhysicsList.hh"

class G4OpWLS;
class G4Cerenkov;
class G4Scintillation;
class G4OpAbsorption;
class G4OpRayleigh;
class G4OpMieHG;
class G4OpBoundaryProcess;

class ScintPhysicsListMessenger;

```

```
//-----------------------------------------------------------------------

class ScintPhysicsList : public G4VUserPhysicsList
{
  public:
    ScintPhysicsList();
   ~ScintPhysicsList();

  public:
    void ConstructParticle();
    void ConstructProcess();

    void SetCuts();

    // These methods construct particles
    void ConstructBosons();
    void ConstructLeptons();
    void ConstructMesons();
    void ConstructBaryons();

    // These methods construct physics processes and register them
    void ConstructGeneral();
    void ConstructEM();
    void ConstructOp();

    // For the messenger class (physicslistmessenger)
    void SetVerbose(G4int);
    void SetNbOfPhotonsCerenkov(G4int);

  private:

    G4OpWLS*              theWLSProcess;
    G4Cerenkov*           theCerenkovProcess;
    G4Scintillation*      theScintillationProcess;
    G4OpAbsorption*       theAbsorptionProcess;
    G4OpRayleigh*         theRayleighScatteringProcess;
    G4OpMieHG*            theMieHGScatteringProcess;
    G4OpBoundaryProcess* theBoundaryProcess;

    ScintPhysicsListMessenger* pMessenger;
};

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

#endif /* ScintPhysicsList_h */
```

./Geant4Code/include/ScintPhysicsList.hh

Code body

```
//
// Physical processes used in simulation
//
//-------------------------------------------------------------------

#include "ScintPhysicsList.hh"
```

```
#include "ScintPhysicsListMessenger.hh"

#include "G4ParticleDefinition.hh"
#include "G4ParticleTypes.hh"
#include "G4ParticleTable.hh"

#include "G4ProcessManager.hh"

#include "G4LossTableManager.hh"
#include "G4EmSaturation.hh"

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

#include "G4MuMultipleScattering.hh"
#include "G4hMultipleScattering.hh"

#include "G4eMultipleScattering.hh"
#include "G4eIonisation.hh"
#include "G4eBremsstrahlung.hh"
#include "G4eplusAnnihilation.hh"

#include "G4MuIonisation.hh"
#include "G4MuBremsstrahlung.hh"
#include "G4MuPairProduction.hh"

#include "G4hIonisation.hh"

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

ScintPhysicsList::ScintPhysicsList() :  G4VUserPhysicsList()
{
  theCerenkovProcess           = NULL;
  theScintillationProcess      = NULL;
  theAbsorptionProcess         = NULL;
  theRayleighScatteringProcess = NULL;
  theMieHGScatteringProcess    = NULL;
  theBoundaryProcess           = NULL;

  pMessenger = new ScintPhysicsListMessenger(this);
  SetVerboseLevel(0);
}

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

ScintPhysicsList::~ScintPhysicsList() { delete pMessenger;}

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

void ScintPhysicsList::ConstructParticle()
{
  // In this method, static member functions should be called
  // for all particles which you want to use.
  // This ensures that objects of these particle types will be
  // created in the program.

  ConstructBosons();
  ConstructLeptons();
  ConstructMesons();
  ConstructBaryons();
}

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

void ScintPhysicsList::ConstructBosons()
{
  // pseudo-particles
  G4Geantino::GeantinoDefinition();
  G4ChargedGeantino::ChargedGeantinoDefinition();

  // gamma
  G4Gamma::GammaDefinition();

  // optical photon
  G4OpticalPhoton::OpticalPhotonDefinition();
}

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

void ScintPhysicsList::ConstructLeptons()
{
  // e+/-
  G4Electron::ElectronDefinition();
  G4Positron::PositronDefinition();

  // mu+/-
  G4MuonPlus::MuonPlusDefinition();
  G4MuonMinus::MuonMinusDefinition();

  // nu_e
  G4NeutrinoE::NeutrinoEDefinition();
  G4AntiNeutrinoE::AntiNeutrinoEDefinition();

  // nu_mu
  G4NeutrinoMu::NeutrinoMuDefinition();
  G4AntiNeutrinoMu::AntiNeutrinoMuDefinition();
}

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

void ScintPhysicsList::ConstructMesons()
{
  // pi +/0/-
  G4PionPlus::PionPlusDefinition();
  G4PionMinus::PionMinusDefinition();
  G4PionZero::PionZeroDefinition();
}

//----------------------------------------------------------------------------
```

```

void ScintPhysicsList::ConstructBaryons()
{
  // p
  G4Proton::ProtonDefinition();
  G4AntiProton::AntiProtonDefinition();

  // n
  G4Neutron::NeutronDefinition();
  G4AntiNeutron::AntiNeutronDefinition();
}

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

void ScintPhysicsList::ConstructProcess()
{
  AddTransportation();
  ConstructGeneral();
  ConstructEM();
  ConstructOp();
}

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

#include "G4Decay.hh"

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

void ScintPhysicsList::ConstructGeneral()
{
  // Add Decay Process
  G4Decay* theDecayProcess = new G4Decay();
  theParticleIterator->reset();
  while( (*theParticleIterator)() ){
    G4ParticleDefinition* particle = theParticleIterator->value();
    G4ProcessManager* pmanager = particle->GetProcessManager();
    if (theDecayProcess->IsApplicable(*particle)) {
      pmanager ->AddProcess(theDecayProcess);
      // set ordering for PostStepDoIt and AtRestDoIt
      pmanager ->SetProcessOrdering(theDecayProcess, idxPostStep);
      pmanager ->SetProcessOrdering(theDecayProcess, idxAtRest);
    }
  }
}

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


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

void ScintPhysicsList::ConstructEM()
{
  theParticleIterator->reset();
  while( (*theParticleIterator)() ){
    G4ParticleDefinition* particle = theParticleIterator->value();
    G4ProcessManager* pmanager = particle->GetProcessManager();
    G4String particleName = particle->GetParticleName();

    if (particleName == "gamma") {
    // gamma
      // Construct processes for gamma
      pmanager->AddDiscreteProcess(new G4GammaConversion()); // pair
   production
      pmanager->AddDiscreteProcess(new G4ComptonScattering());
      pmanager->AddDiscreteProcess(new G4PhotoElectricEffect());

    } else if (particleName == "e-") {
    //electron
      // Construct processes for electron
      pmanager->AddProcess(new G4eMultipleScattering(),-1, 1, 1);
      pmanager->AddProcess(new G4eIonisation(),        -1, 2, 2);
      pmanager->AddProcess(new G4eBremsstrahlung(),    -1, 3, 3);

    } else if (particleName == "e+") {
    //positron
      // Construct processes for positron
      pmanager->AddProcess(new G4eMultipleScattering(),-1, 1, 1);
      pmanager->AddProcess(new G4eIonisation(),        -1, 2, 2);
      pmanager->AddProcess(new G4eBremsstrahlung(),    -1, 3, 3);
      pmanager->AddProcess(new G4eplusAnnihilation(),  0,-1, 4);

    } else if( particleName == "mu+" ||
               particleName == "mu-"    ) {
    //muon
     // Construct processes for muon
     pmanager->AddProcess(new G4MuMultipleScattering(),-1, 1, 1);
     pmanager->AddProcess(new G4MuIonisation(),       -1, 2, 2);
     pmanager->AddProcess(new G4MuBremsstrahlung(),   -1, 3, 3);
     pmanager->AddProcess(new G4MuPairProduction(),   -1, 4, 4);

    } else {
      if ((particle->GetPDGCharge() != 0.0) &&
          (particle->GetParticleName() != "chargedgeantino")) {
       // all others charged particles except geantino
       pmanager->AddProcess(new G4hMultipleScattering(),-1,1,1);
       pmanager->AddProcess(new G4hIonisation(),        -1,2,2);
     }
    }
  }
}

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

void ScintPhysicsList::ConstructOp()
{

  G4cout << "WLSOpticalPhysics:: Add Optical Physics Processes"
           << G4endl;

  theWLSProcess = new G4OpWLS();

  theScintillationProcess = new G4Scintillation();
  theScintillationProcess->SetScintillationYieldFactor(1.);
  theScintillationProcess->SetTrackSecondariesFirst(true);

  theCerenkovProcess = new G4Cerenkov();
  theCerenkovProcess->SetMaxNumPhotonsPerStep(300);
  theCerenkovProcess->SetTrackSecondariesFirst(true);

  theAbsorptionProcess           = new G4OpAbsorption();
  theRayleighScatteringProcess   = new G4OpRayleigh();
  theMieHGScatteringProcess      = new G4OpMieHG();
  theBoundaryProcess             = new G4OpBoundaryProcess();

  G4ProcessManager* pManager =
                G4OpticalPhoton::OpticalPhoton()->GetProcessManager();

  if (!pManager) {
     std::ostringstream o;
     o << "Optical Photon without a Process Manager";
     G4Exception("WLSOpticalPhysics::ConstructProcess()","",
                 FatalException,o.str().c_str());
  }

  pManager->AddDiscreteProcess(theAbsorptionProcess);

  //pManager->AddDiscreteProcess(theRayleighScattering);
  //pManager->AddDiscreteProcess(theMieHGScatteringProcess);

  pManager->AddDiscreteProcess(theBoundaryProcess);

  theWLSProcess->UseTimeProfile("delta");
  //theWLSProcess->UseTimeProfile("exponential");

  pManager->AddDiscreteProcess(theWLSProcess);

  theScintillationProcess->SetScintillationYieldFactor(1.);
  theScintillationProcess->SetScintillationExcitationRatio(0.0);
  theScintillationProcess->SetTrackSecondariesFirst(true);

  // Use Birks Correction in the Scintillation process

  G4EmSaturation* emSaturation = G4LossTableManager::Instance()->
    EmSaturation();
  theScintillationProcess->AddSaturation(emSaturation);

  theParticleIterator->reset();
  while ( (*theParticleIterator)() ){

    G4ParticleDefinition* particle = theParticleIterator->value();
    G4String particleName = particle->GetParticleName();

    pManager = particle->GetProcessManager();
    if (!pManager) {
       std::ostringstream o;
```

```
        o << "Particle " << particleName << "without a Process Manager";
        G4Exception("WLSOpticalPhysics::ConstructProcess()","",
                    FatalException,o.str().c_str());
     }

     if(theCerenkovProcess->IsApplicable(*particle)){
       pManager->AddProcess(theCerenkovProcess);
       pManager->SetProcessOrdering(theCerenkovProcess,idxPostStep);
     }
     if(theScintillationProcess->IsApplicable(*particle)){
       pManager->AddProcess(theScintillationProcess);
       pManager->SetProcessOrderingToLast(theScintillationProcess,
    idxAtRest);
       pManager->SetProcessOrderingToLast(theScintillationProcess,
    idxPostStep);
     }

  }


  theWLSProcess                  = new G4OpWLS();
  theCerenkovProcess             = new G4Cerenkov("Cerenkov");
  theScintillationProcess        = new G4Scintillation("Scintillation");
  theAbsorptionProcess           = new G4OpAbsorption();
  theRayleighScatteringProcess   = new G4OpRayleigh();
  theMieHGScatteringProcess      = new G4OpMieHG();
  theBoundaryProcess             = new G4OpBoundaryProcess();

//   theCerenkovProcess->DumpPhysicsTable();
//   theScintillationProcess->DumpPhysicsTable();
//   theRayleighScatteringProcess->DumpPhysicsTable();

  //SetVerbose(1);

  theCerenkovProcess->SetMaxNumPhotonsPerStep(300); // max AVERAGE #
    photons per step (limits step size to match this average).
  theCerenkovProcess->SetMaxBetaChangePerStep(10.0);
  theCerenkovProcess->SetTrackSecondariesFirst(true);

  theScintillationProcess->SetScintillationYieldFactor(1.);
  theScintillationProcess->SetScintillationExcitationRatio(0.0);
  theScintillationProcess->SetTrackSecondariesFirst(true);

  theWLSProcess->UseTimeProfile("exponential");

  // Use Birks Correction in the Scintillation process

  G4EmSaturation* emSaturation = G4LossTableManager::Instance()->
    EmSaturation();
  theScintillationProcess->AddSaturation(emSaturation);

  theParticleIterator->reset();
  while( (*theParticleIterator)() )
  {
    G4ParticleDefinition* particle = theParticleIterator->value();
     G4ProcessManager* pmanager = particle->GetProcessManager();
     G4String particleName = particle->GetParticleName();
     if (theCerenkovProcess->IsApplicable(*particle)) {
       pmanager->AddProcess(theCerenkovProcess);
       pmanager->SetProcessOrdering(theCerenkovProcess,idxPostStep);
     }
     if (theScintillationProcess->IsApplicable(*particle)) {
       pmanager->AddProcess(theScintillationProcess);
       pmanager->SetProcessOrderingToLast(theScintillationProcess,
    idxAtRest);
       pmanager->SetProcessOrderingToLast(theScintillationProcess,
    idxPostStep);
     }
     if (theWLSProcess->IsApplicable(*particle)) {
       pmanager->AddDiscreteProcess(theWLSProcess);
     }
     if (particleName == "opticalphoton") {
       G4cout << " AddDiscreteProcess to OpticalPhoton " << G4endl;
       pmanager->AddDiscreteProcess(theWLSProcess);
       pmanager->AddDiscreteProcess(theAbsorptionProcess);
       pmanager->AddDiscreteProcess(theRayleighScatteringProcess);
       pmanager->AddDiscreteProcess(theMieHGScatteringProcess);
       pmanager->AddDiscreteProcess(theBoundaryProcess);
     }
  }
}

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

void ScintPhysicsList::SetVerbose(G4int verbose)
{
  theCerenkovProcess->SetVerboseLevel(verbose);
  theScintillationProcess->SetVerboseLevel(verbose);
  theAbsorptionProcess->SetVerboseLevel(verbose);
  theRayleighScatteringProcess->SetVerboseLevel(verbose);
  theMieHGScatteringProcess->SetVerboseLevel(verbose);
  theBoundaryProcess->SetVerboseLevel(verbose);
  theWLSProcess->SetVerboseLevel(1);
}

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

void ScintPhysicsList::SetNbOfPhotonsCerenkov(G4int MaxNumber)
{
  theCerenkovProcess->SetMaxNumPhotonsPerStep(MaxNumber);
}
//-----------------------------------------------------------------

void ScintPhysicsList::SetCuts()
{
  //  " G4VUserPhysicsList::SetCutsWithDefault" method sets
  //   the default cut value for all particle types
  //
  SetCutsWithDefault();

  if (verboseLevel>0) DumpCutValuesTable();
}

//----------------------------------------------------------------------------
```

./Geant4Code/src/ScintPhysicsList.cc

A.3.3 Physics List Messenger

Header file

```
//
// Facilities delivering info from physics list
//

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

#ifndef ScintPhysicsListMessenger_h
#define ScintPhysicsListMessenger_h 1

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

class ScintPhysicsList;
class G4UIdirectory;
class G4UIcmdWithAnInteger;

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

class ScintPhysicsListMessenger: public G4UImessenger
{
  public:
    ScintPhysicsListMessenger(ScintPhysicsList* );
   ~ScintPhysicsListMessenger();

    void SetNewValue(G4UIcommand*, G4String);

  private:
    ScintPhysicsList*      pPhysicsList;

    G4UIdirectory*         ScintDir;
    G4UIdirectory*         physDir;
    G4UIcmdWithAnInteger*  verboseCmd;
    G4UIcmdWithAnInteger*  cerenkovCmd;
};

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

#endif
```

./Geant4Code/include/ScintPhysicsListMessenger.hh

Code body

```
//
```

```
//

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

#include "ScintPhysicsListMessenger.hh"

#include "ScintPhysicsList.hh"
#include "G4UIdirectory.hh"
#include "G4UIcmdWithAnInteger.hh"

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

ScintPhysicsListMessenger::ScintPhysicsListMessenger(ScintPhysicsList*
    pPhys)
:pPhysicsList(pPhys)
{
  ScintDir = new G4UIdirectory("/Scint/");
  ScintDir->SetGuidance("UI commands of this example");

  physDir = new G4UIdirectory("/Scint/phys/");
  physDir->SetGuidance("PhysicsList control");

  verboseCmd = new G4UIcmdWithAnInteger("/Scint/phys/verbose",this);
  verboseCmd->SetGuidance("set verbose for physics processes");
  verboseCmd->SetParameterName("verbose",true);
  verboseCmd->SetDefaultValue(1);
  verboseCmd->SetRange("verbose>=0");
  verboseCmd->AvailableForStates(G4State_PreInit,G4State_Idle);

  cerenkovCmd = new G4UIcmdWithAnInteger("/Scint/phys/cerenkovMaxPhotons
    ",this);
  cerenkovCmd->SetGuidance("set max nb of photons per step");
  cerenkovCmd->SetParameterName("MaxNumber",false);
  cerenkovCmd->SetRange("MaxNumber>=0");
  cerenkovCmd->AvailableForStates(G4State_Idle);
}

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

ScintPhysicsListMessenger::~ScintPhysicsListMessenger()
{
  delete verboseCmd;
  delete cerenkovCmd;
  delete physDir;
  delete ScintDir;
}

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

void ScintPhysicsListMessenger::SetNewValue(G4UIcommand* command,
                                            G4String newValue)
{
  if( command == verboseCmd )
   { pPhysicsList->SetVerbose(verboseCmd->GetNewIntValue(newValue));}

  if( command == cerenkovCmd )
   {pPhysicsList->SetNbOfPhotonsCerenkov(cerenkovCmd->GetNewIntValue(
    newValue));}
}

//----------------------------------------------------------------------
```

./Geant4Code/src/ScintPhysicsListMessenger.cc

A.3.4 Primary Action Generator

Header file

```
/*
 * Written by: Nathan Murtha
 * December 18, 2014
 *
 * What is needed for the primary particle generation
 *
 */


#ifndef ScintPrimaryGeneratorAction_h
#define ScintPrimaryGeneratorAction_h 1

#include "G4VUserPrimaryGeneratorAction.hh"
#include "ScintPrimaryGeneratorMessenger.hh"

#include "G4ParticleGun.hh"
#include "globals.hh"


class G4ParticleGun;
class G4Event;
class ScintDetectorConstruction;

class ScintPrimaryGeneratorAction : public G4VUserPrimaryGeneratorAction
{
    public:
    ScintPrimaryGeneratorAction();
    virtual ~ScintPrimaryGeneratorAction();

    // Method from the base class
    virtual void GeneratePrimaries(G4Event*);

    void SetOptPhotonPolar();
    void SetOptPhotonPolar(G4double);

    private:
    G4ParticleGun*  ParticleGun; // Pointer a to G4 particle gun class
    ScintPrimaryGeneratorMessenger* gunMessenger;
};

#endif
```

./Geant4Code/include/ScintPrimaryGeneratorAction.hh

Code body

```
/*
 * Written by: Nathan Murtha
 * December 18, 2014
 *
 * Generates primary events for ”ScintScintillatorTest”
 *
 */

#include ”ScintPrimaryGeneratorAction.hh”
#include ”ScintPrimaryGeneratorMessenger.hh”

#include ”Randomize.hh”

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

ScintPrimaryGeneratorAction::ScintPrimaryGeneratorAction()
 : G4VUserPrimaryGeneratorAction(),
   ParticleGun(0)
{
  // Initialize particle gun
    G4int n_particle = 1;
    ParticleGun = new G4ParticleGun(n_particle);

  // create a messenger for this class
    gunMessenger = new ScintPrimaryGeneratorMessenger(this);

  // Get particle table, in order to look up particles to generate
    G4ParticleTable* ParticleTable = G4ParticleTable::GetParticleTable();

  // Get particle to shoot (in this case a muon)
    G4String particleName;
    G4ParticleDefinition* particle = ParticleTable->FindParticle(particleName=”opticalphoton”);

  // Set particle gun settings
    ParticleGun->SetParticleDefinition(particle);
    ParticleGun->SetParticleTime(0.0*ns);
    ParticleGun->SetParticlePosition(G4ThreeVector(-1.265*cm, 10.0*cm, -37.6*cm));
    ParticleGun->SetParticleMomentumDirection(G4ThreeVector(0.0,-1.0,1.0));
    ParticleGun->SetParticleEnergy(2.92*eV);
}

ScintPrimaryGeneratorAction::~ScintPrimaryGeneratorAction()
```

```
{
  delete ParticleGun;
  delete gunMessenger;
}

void ScintPrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{
    ParticleGun->GeneratePrimaryVertex(anEvent);
}

void ScintPrimaryGeneratorAction::SetOptPhotonPolar()
{
 G4double angle = G4UniformRand() * 360.0*deg;
 SetOptPhotonPolar(angle);
}


void ScintPrimaryGeneratorAction::SetOptPhotonPolar(G4double angle)
{
 if (ParticleGun->GetParticleDefinition()->GetParticleName() != "
    opticalphoton")
   {
     G4cout << "--> warning from PrimaryGeneratorAction::
    SetOptPhotonPolar() :"
               "the ParticleGun is not an opticalphoton" << G4endl;
     return;
   }

 G4ThreeVector normal (1., 0., 0.);
 G4ThreeVector kphoton = ParticleGun->GetParticleMomentumDirection();
 G4ThreeVector product = normal.cross(kphoton);
 G4double modul2       = product*product;

 G4ThreeVector e_perpend (0., 0., 1.);
 if (modul2 > 0.) e_perpend = (1./std::sqrt(modul2))*product;
 G4ThreeVector e_paralle    = e_perpend.cross(kphoton);

 G4ThreeVector polar = std::cos(angle)*e_paralle + std::sin(angle)*
    e_perpend;
 ParticleGun->SetParticlePolarization(polar);
}
```

./Geant4Code/src/ScintPrimaryGeneratorAction.cc

A.3.5 Primary Generator Messenger

Header file

```
//
// Facilitates delivering info from primary generator
//
//
//

//---------------------------------------------------------------------
```

```

#ifndef ScintPrimaryGeneratorMessenger_h
#define ScintPrimaryGeneratorMessenger_h 1


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

class ScintPrimaryGeneratorAction;
class G4UIdirectory;
class G4UIcmdWithADoubleAndUnit;

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

class ScintPrimaryGeneratorMessenger: public G4UImessenger
{
  public:
    ScintPrimaryGeneratorMessenger(ScintPrimaryGeneratorAction*);
   ~ScintPrimaryGeneratorMessenger();

    void SetNewValue(G4UIcommand*, G4String);

  private:
    ScintPrimaryGeneratorAction* ScintAction;
    G4UIdirectory*               gunDir;
    G4UIcmdWithADoubleAndUnit*   polarCmd;
};

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

#endif
```

./Geant4Code/include/ScintPrimaryGeneratorMessenger.hh

Code body

```
//
//

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

#include "ScintPrimaryGeneratorMessenger.hh"
#include "ScintPrimaryGeneratorAction.hh"

#include "G4UIdirectory.hh"
#include "G4UIcmdWithADoubleAndUnit.hh"
#include "G4SystemOfUnits.hh"

//-----------------------------------------------------------------------------
ScintPrimaryGeneratorMessenger::ScintPrimaryGeneratorMessenger(
                                            ScintPrimaryGeneratorAction*
    ScintGun)
:ScintAction(ScintGun)
{
  gunDir = new G4UIdirectory("/Scint/gun/");
  gunDir->SetGuidance("PrimaryGenerator control");

  polarCmd = new G4UIcmdWithADoubleAndUnit("/Scint/gun/optPhotonPolar",
    this);
  polarCmd->SetGuidance("Set linear polarization");
  polarCmd->SetGuidance("  angle w.r.t. (k,n) plane");
  polarCmd->SetParameterName("angle",true);
  polarCmd->SetUnitCategory("Angle");
  polarCmd->SetDefaultValue(-360.0);
  polarCmd->SetDefaultUnit("deg");
  polarCmd->AvailableForStates(G4State_Idle);
}

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

ScintPrimaryGeneratorMessenger::~ScintPrimaryGeneratorMessenger()
{
  delete polarCmd;
  delete gunDir;
}

//---------------------------------------------------------------------
void ScintPrimaryGeneratorMessenger::SetNewValue(
                                            G4UIcommand* command, G4String
   newValue)
{
  if( command == polarCmd ) {
      G4double angle = polarCmd->GetNewDoubleValue(newValue);
      //if ( angle == -360.0*deg ) {
      //    ScintAction->SetOptPhotonPolar();
      //} else {
        // ScintAction->SetOptPhotonPolar(angle);
      //}
  }
}
//---------------------------------------------------------------------
```

./Geant4Code/src/ScintPrimaryGeneratorMessenger.cc

A.3.6 Run Action

Header file

```
//
//
// Contains info for each run
//
//-------------------------------------------------------------------------

#ifndef ScintRunAction_h
#define ScintRunAction_h 1

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

//-------------------------------------------------------------------------
```

```

class G4Timer;
class G4Run;

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

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

  private:
    G4Timer* timer;
};

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

#endif /*ScintRunAction_h*/
```

./Geant4Code/include/ScintRunAction.hh

Code body

```
//
//------------------------------------------------------------------

// Make this appear first!
#include "G4Timer.hh"

#include "ScintRunAction.hh"

#include "G4Run.hh"

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

ScintRunAction::ScintRunAction()
{
  timer = new G4Timer;
}

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

ScintRunAction::~ScintRunAction()
{
  delete timer;
}

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

void ScintRunAction::BeginOfRunAction(const G4Run* aRun)
{
  G4cout << "### Run " << aRun->GetRunID() << " start." << G4endl;
  timer->Start();
}

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

void ScintRunAction::EndOfRunAction(const G4Run* aRun)
{
  timer->Stop();
  G4cout << "number of event = " << aRun->GetNumberOfEvent()
         << " " << *timer << G4endl;
}

//----------------------------------------------------------------------
```

./Geant4Code/src/ScintRunAction.cc

A.3.7 Sensitive Detectors

Header file

```
//
//
// Contains the pieces needed for a sensitive detector
//
#ifndef ScintSensitiveDetector_h
#define ScintSensitiveDetector_h 1

#include "G4VSensitiveDetector.hh"

class G4Step;

class ScintSensitiveDetector: public G4VSensitiveDetector
{
    private:
    public:
        ScintSensitiveDetector(G4String name);
        ~ScintSensitiveDetector();
        G4bool ProcessHits(G4Step *step, G4TouchableHistory *hist);
        void EndOfEvent(G4HCofThisEvent*);

        G4double GetEnergy(G4Step *step);
};

#endif
```

./Geant4Code/include/ScintSensitiveDetector.hh

Code body

```
/*
 * Written by Nathan Murtha
 * December 18, 2014
 *
 *
 *
 */
```

```

#include "G4Step.hh"
#include <fstream>
#include <iostream>
#include "stdlib.h"
#include "ScintSensitiveDetector.hh"

using namespace std;

ScintSensitiveDetector::ScintSensitiveDetector(G4String name)
 : G4VSensitiveDetector(name)
{
  std::ofstream file("outputScintTest.dat");
  file << "Photons that made it to the end of the fiber..." << endl;
}

G4bool ScintSensitiveDetector::ProcessHits(G4Step *step,
    G4TouchableHistory *hist)
{

  G4double energy = step->GetTrack()->GetDynamicParticle()->
    GetKineticEnergy();
  //G4double x = step->GetTrack()->GetPosition().getX();
  //G4double y = step->GetTrack()->GetPosition().getY();
  //G4double z = step->GetTrack()->GetPosition().getZ();
  //G4int PDGcode = step->GetTrack()->GetDynamicParticle()->
    GetParticleDefinition()->GetPDGEncoding();
  G4String particle = step->GetTrack()->GetDynamicParticle()->
    GetParticleDefinition()->GetParticleName();

  step->GetTrack()->SetTrackStatus(fStopAndKill);

  std::ofstream file("outputScintTest.dat",ios::app); //append to file

  //file.setf(ios::fixed,ios::floatfield);
  //cout.precision(4);

  if (particle == "opticalphoton")
  {
     file << energy/eV << endl;
  }

/*
  file << std::setw(10) << energy/MeV << " "
       << std::setw(10) << x/cm << " "
       << std::setw(10) << y/cm << " "
       << std::setw(10) << z/cm << " "
       << std::setw(10) << PDGcode << " "
       << std::endl;

  file << setprecision(4) << particle;
  file << "\t\t" << setprecision(4) << energy;
  file << "\t\t" << setprecision(4) << x;
  file << "\t\t" << setprecision(4) << y;
  file << "\t\t" << setprecision(4) << z;
  file << "\t\t" << PDGcode << endl;

*/

  return true;
}


void ScintSensitiveDetector::EndOfEvent(G4HCofThisEvent*)
{
}



ScintSensitiveDetector::~ScintSensitiveDetector()
{
}
```

./Geant4Code/src/ScintSensitiveDetector.cc

A.3.8 Stacking Action

Header file

```
//
// Contains info to stack particle tracks
//
//—————————————————————————————————————————————

#ifndef ScintStackingAction_H
#define ScintStackingAction_H 1

#include "globals.hh"
#include "G4UserStackingAction.hh"

//—————————————————————————————————————————————

class ScintStackingAction : public G4UserStackingAction
{
  public:
    ScintStackingAction();
   ~ScintStackingAction();

  public:
    G4ClassificationOfNewTrack ClassifyNewTrack(const G4Track* aTrack);
    void NewStage();
    void PrepareNewEvent();

  private:
    G4int photonCounter;
};

//—————————————————————————————————————————————

#endif
```

./Geant4Code/include/ScintStackingAction.hh

Code body

```
//
//----------------------------------------------------------------------------

#include "ScintStackingAction.hh"

#include "G4ParticleDefinition.hh"
#include "G4ParticleTypes.hh"
#include "G4Track.hh"
#include "G4ios.hh"

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

ScintStackingAction::ScintStackingAction()
: photonCounter(0)
{}

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

ScintStackingAction::~ScintStackingAction()
{}

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

G4ClassificationOfNewTrack
ScintStackingAction::ClassifyNewTrack(const G4Track * aTrack)
{
  if(aTrack->GetDefinition() == G4OpticalPhoton::OpticalPhotonDefinition
    ())
  { // particle is optical photon
    if(aTrack->GetParentID()>0)
    { // particle is secondary
      photonCounter++;
    }
  }
  return fUrgent;
}

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

void ScintStackingAction::NewStage()
{
  G4cout << "Number of optical photons produced in this event : "
         << photonCounter << G4endl;
}

//----------------------------------------------------------------------------
void ScintStackingAction::PrepareNewEvent()
{ photonCounter = 0; }

//----------------------------------------------------------------------------
```

./Geant4Code/src/ScintStackingAction.cc

A.3.9 Stepping Action

Header file

```
/*
 * Written by: Nathan Murtha
 * December 18, 2014
 *
 * Contains the basics needed for energy deposition calculation
 *
 */

#ifndef ScintSteppingAction_h
#define ScintSteppingAction_h 1

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

class G4LogicalVolume;

// Stepping action class
//
// It holds data member fEnergy for accumulating the energy deposit
// in a selected volume step by step. The selected volume is set from
// the detector construction via the SetVolume() function. The
// accumulated energy deposit is reset for each new event via
// the Reset() function from the event action.

class ScintSteppingAction : public G4UserSteppingAction
{
  public:
    ScintSteppingAction();
    virtual ~ScintSteppingAction();

    // static access method
    static ScintSteppingAction* Instance();

    // method from the base class
    virtual void UserSteppingAction(const G4Step*);

    // reset accumulated energy
    void Reset();

    // set methods
    void SetVolume(G4LogicalVolume* volume) { fVolume = volume; }

    // get methods
    G4LogicalVolume* GetVolume() const { return fVolume; }
    G4double GetEnergy() const { return fEnergy; }

  private:
    static ScintSteppingAction* fgInstance;

    G4LogicalVolume* fVolume;
    G4double  fEnergy;
};

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

#endif
```

./Geant4Code/include/ScintSteppingAction.hh

Code body

```
/*
 * Written by Nathan Murtha
 * December 18, 2014
 *
 * Contains the basics needed for energy deposition calculation
 *
 */

#include "ScintSteppingAction.hh"

#include "ScintDetectorConstruction.hh"

#include "G4Step.hh"
#include "G4RunManager.hh"
#include "G4UnitsTable.hh"

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

ScintSteppingAction* ScintSteppingAction::fgInstance = 0;

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

ScintSteppingAction* ScintSteppingAction::Instance()
{
// Static acces function via G4RunManager

  return fgInstance;
}

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

ScintSteppingAction::ScintSteppingAction()
: G4UserSteppingAction(),
  fVolume(0),
  fEnergy(0.)
{
  fgInstance = this;
}

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

ScintSteppingAction::~ScintSteppingAction()
```

```
{
  fgInstance = 0;
}

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

void ScintSteppingAction::UserSteppingAction(const G4Step* step)
{
  // get volume of the current step
  G4LogicalVolume* volume
    = step->GetPreStepPoint()->GetTouchableHandle()
      ->GetVolume()->GetLogicalVolume();

  // check if we are in scoring volume
  if (volume != fVolume ) return;

  // collect energy and track length step by step
  G4double edep = step->GetTotalEnergyDeposit();
  fEnergy += edep;
}

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

void ScintSteppingAction::Reset()
{
  fEnergy = 0.;
}
```

./Geant4Code/src/ScintSteppingAction.cc

A.3.10 Stepping Verbosity

Header file

```
//
// Contains info for displaying particle track data
//
//----------------------------------------------------------------------

class ScintSteppingVerbose;

#ifndef ScintSteppingVerbose_h
#define ScintSteppingVerbose_h 1

#include "G4SteppingVerbose.hh"

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

class ScintSteppingVerbose : public G4SteppingVerbose
{
 public:

   ScintSteppingVerbose();
  ~ScintSteppingVerbose();

```

```
   void StepInfo();
   void TrackingStarted();

};

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

#endif
```

./Geant4Code/include/ScintSteppingVerbose.hh

Code body

```
//
//-----------------------------------------------------------------

#include "ScintSteppingVerbose.hh"

#include "G4SteppingManager.hh"
#include "G4UnitsTable.hh"

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

ScintSteppingVerbose::ScintSteppingVerbose()
{}

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

ScintSteppingVerbose::~ScintSteppingVerbose()
{}

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

void ScintSteppingVerbose::StepInfo()
{
  CopyState();

  G4int prec = G4cout.precision(3);

  if( verboseLevel >= 1 ){
    if( verboseLevel >= 4 ) VerboseTrack();
    if( verboseLevel >= 3 ){
      G4cout << G4endl;
      G4cout << std::setw( 5) << "#Step#"     << " "
             << std::setw( 6) << "X"          << "    "
             << std::setw( 6) << "Y"          << "    "
             << std::setw( 6) << "Z"          << "    "
             << std::setw( 9) << "KineE"      << " "
             << std::setw( 9) << "dEStep"     << " "
             << std::setw(10) << "StepLeng"
             << std::setw(10) << "TrakLeng"
             << std::setw(10) << "Volume"    << "  "
             << std::setw(10) << "Process"   << G4endl;
    }

    G4cout << std::setw(5) << fTrack->GetCurrentStepNumber() << " "
```

```
   << std::setw(6) << G4BestUnit(fTrack->GetPosition().x(),"Length")
   << std::setw(6) << G4BestUnit(fTrack->GetPosition().y(),"Length")
   << std::setw(6) << G4BestUnit(fTrack->GetPosition().z(),"Length")
   << std::setw(6) << G4BestUnit(fTrack->GetKineticEnergy(),"Energy")
   << std::setw(6) << G4BestUnit(fStep->GetTotalEnergyDeposit(),"Energy
  ")
   << std::setw(6) << G4BestUnit(fStep->GetStepLength(),"Length")
   << std::setw(6) << G4BestUnit(fTrack->GetTrackLength(),"Length")
   << "  ";

   // if( fStepStatus != fWorldBoundary){
   if( fTrack->GetNextVolume() != 0 ) {
     G4cout << std::setw(10) << fTrack->GetVolume()->GetName();
   } else {
     G4cout << std::setw(10) << "OutOfWorld";
   }

   if(fStep->GetPostStepPoint()->GetProcessDefinedStep() != 0){
     G4cout << "  "
            << std::setw(10)
            << fStep->GetPostStepPoint()->GetProcessDefinedStep()
                                              ->GetProcessName();
   } else {
     G4cout << "   UserLimit";
   }

   G4cout << G4endl;

   if( verboseLevel == 2 ){
     G4int tN2ndariesTot = fN2ndariesAtRestDoIt +
                           fN2ndariesAlongStepDoIt +
                           fN2ndariesPostStepDoIt;
     if(tN2ndariesTot>0){
       G4cout << "    :----- List of 2ndaries - "
              << "#SpawnInStep=" << std::setw(3) << tN2ndariesTot
              << "(Rest="  << std::setw(2) << fN2ndariesAtRestDoIt
              << ",Along=" << std::setw(2) << fN2ndariesAlongStepDoIt
              << ",Post="  << std::setw(2) << fN2ndariesPostStepDoIt
              << "), "
              << "#SpawnTotal=" << std::setw(3) << (*fSecondary).size()
              << " ---------------"
              << G4endl;

       for(size_t lp1=(*fSecondary).size()-tN2ndariesTot;
                   lp1<(*fSecondary).size(); lp1++){
         G4cout << "     : "
                << std::setw(6)
                << G4BestUnit((*fSecondary)[lp1]->GetPosition().x(),"
  Length")
                << std::setw(6)
                << G4BestUnit((*fSecondary)[lp1]->GetPosition().y(),"
  Length")
                << std::setw(6)
                << G4BestUnit((*fSecondary)[lp1]->GetPosition().z(),"
  Length")
```

```
                    << std::setw(6)
                    << G4BestUnit((*fSecondary)[lp1]->GetKineticEnergy(),"
    Energy")
                    << std::setw(10)
                    << (*fSecondary)[lp1]->GetDefinition()->GetParticleName
    ();
            G4cout << G4endl;
          }

          G4cout << "     :-----------------------------"
                 << "-------------------------------------"
                 << "-- EndOf2ndaries Info ---------------"
                 << G4endl;
        }
      }

  }
  G4cout.precision(prec);
}

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

void ScintSteppingVerbose::TrackingStarted()
{

  CopyState();
G4int prec = G4cout.precision(3);
  if( verboseLevel > 0 ){

    G4cout << std::setw( 5) << "Step#"      << " "
           << std::setw( 6) << "X"          << "    "
           << std::setw( 6) << "Y"          << "    "
           << std::setw( 6) << "Z"          << "    "
           << std::setw( 9) << "KineE"      << " "
           << std::setw( 9) << "dEStep"     << " "
           << std::setw(10) << "StepLeng"
           << std::setw(10) << "TrakLeng"
           << std::setw(10) << "Volume"     << "  "
           << std::setw(10) << "Process"    << G4endl;

   G4cout << std::setw( 5) << fTrack->GetCurrentStepNumber() << " "
    << std::setw( 6) << G4BestUnit(fTrack->GetPosition().x(),"Length")
    << std::setw( 6) << G4BestUnit(fTrack->GetPosition().y(),"Length")
    << std::setw( 6) << G4BestUnit(fTrack->GetPosition().z(),"Length")
    << std::setw( 6) << G4BestUnit(fTrack->GetKineticEnergy(),"Energy")
    << std::setw( 6) << G4BestUnit(fStep->GetTotalEnergyDeposit(),"
    Energy")
    << std::setw( 6) << G4BestUnit(fStep->GetStepLength(),"Length")
    << std::setw( 6) << G4BestUnit(fTrack->GetTrackLength(),"Length")
    << "  ";

    if(fTrack->GetNextVolume()){
      G4cout << std::setw(10) << fTrack->GetVolume()->GetName();
    } else {
      G4cout << "OutOfWorld";
    }
    G4cout << "     initStep" << G4endl;
  }
  G4cout.precision(prec);
}

//--------------------------------------------------------------------------------
```

./Geant4Code/src/ScintSteppingVerbose.cc