Energy Deposition in Polymers

Matthew J. Urffer

TODO LIST

Contents

List of Figures

LIST OF TABLES

LISTINGS

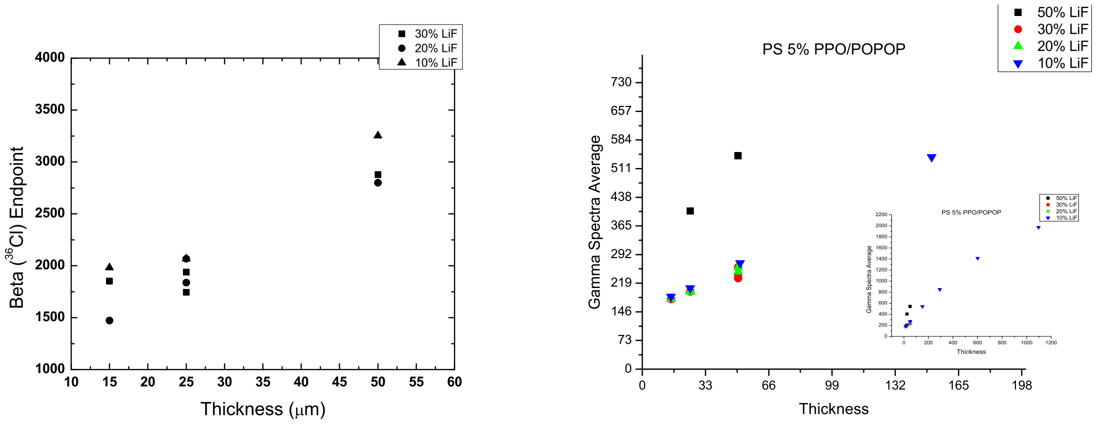
17 World Physical Volume . . . 28
18 Calorimeter Volume . . . 29
19 Layer Volume . . . 29
20 Absorber and Gap Volumes . . . 29
21 Implemented Physics List . . . 31
22 Implemented Physics List . . . 32
23 Primary Event Generator . . . 32
24 Generate Primaries . . . 33
25 Calorimeter Hit . . . 33
26 Sensitive Detector . . . 35
27 Creating Sensitive Detectors . . . 36
28 Event Action . . . 36
29 Run Action . . . 37
30 Process Hit Collection . . . 37
31 Run Macro . . . 39

I. Previous Work

Previous work on the energy deposition of thin films focused on spectra measurements from fabricated films along with single collision energy loss spectra for physical insights. A sequence of 10% ^{6}LiF, 5% PPO-POPOP films in a PS matrix cast to thickness between 15 and 600 μm were fabricated and the response was measured from a gamma source as well as a neutron source. These experiment results are shown in I-A. The single collision energy loss spectra was investigated for electrons in water in order to provide insight on the amount of energy an electron loses in a collision. These results are discussed in Section I-B.

A. Spectra Measurements

Evidence that the secondary electrons contribute to energy loss can be seen in Figure 1 where there is an increase in the endpoint of the spectra as films become thicker. This increase in the spectra endpoint is indicative of the film producing more light, and as the light collection geometry remained constant, the increase in the endpoint is attributed to a larger energy deposition in the 50 μm film compared to the 15 μm or 25 μm film. Figure 2 shows



(a) Beta (^{36}Cl) Spectra Endpoints for PS films

(b) Gamma (^{60}Co) Spectra Averages for PS films

Fig. 1: Spectra properties as a function of film thickness

the intrinsic efficiency of these film from spectra obtained from a ^{60}Co source. As the film thickness increases the pulse height discriminator at which an intrinsic efficiency of one in a million ($\epsilon_{int,\gamma} \leq 10^{-6}$) is reached also increases. The neutron spectra (shown in the solid lines) does not increase in light yield with increasing thickness, further providing an indication that the thickness of the films can be optimized to maximize the neutron count rates[1] while minimizing the response of the detector to photons.

[1]The neutron count rate is increased with thickness by the increased mass of the detector

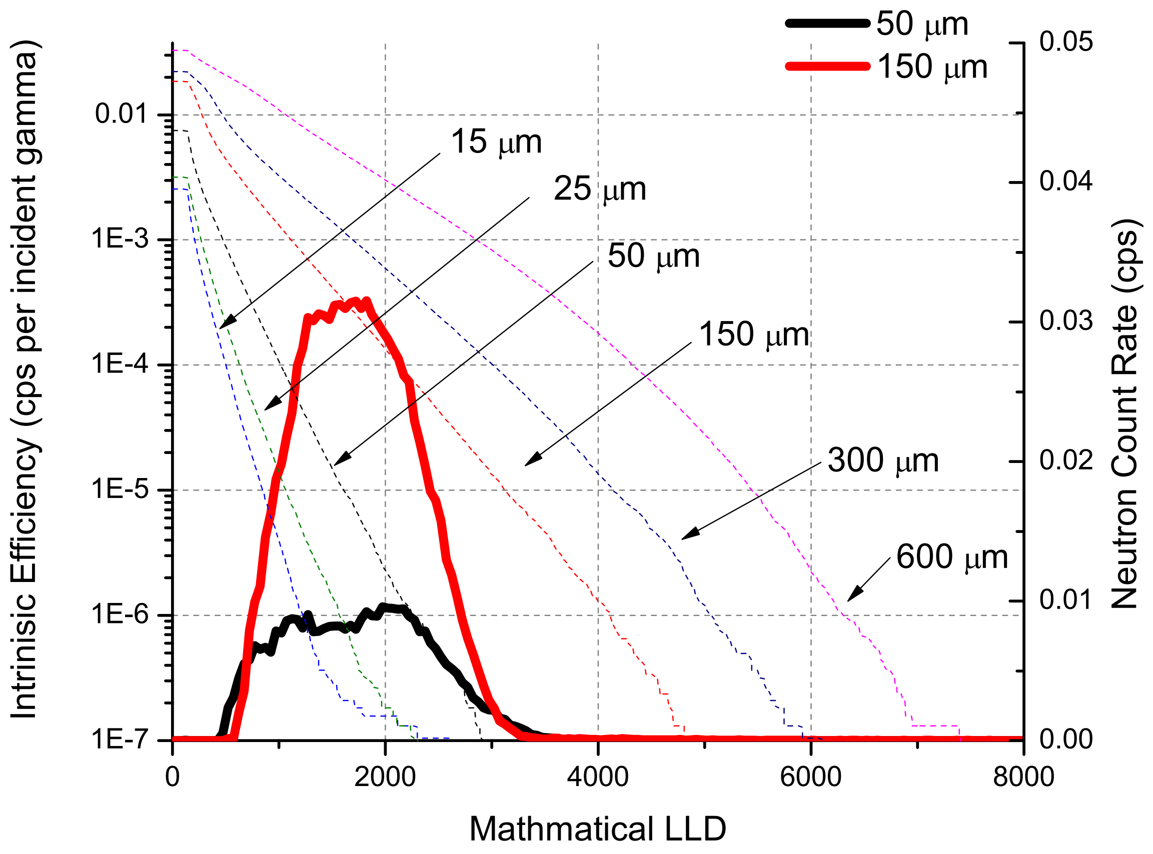


Fig. 2: Gamma intrinsic efficiency (dashed lines) plotted against neutron counts (solid)

B. Single Collision Energy Loss

Single collision energy loss spectra provides the probability that that a given collision will result in an energy loss. Provided a spectra of secondary electrons from either the Compton scattered electron or the ^{6}Li reaction products it is then possible to determine the average energy loss per collision. A single collision energy loss spectra for water is shown in Figure 3. For low electron energies ($<$ 50 eV) it is very probable that the electron will lose a majority of its energy in a single collision. More energetic electrons, however, tend to lose a lower fraction of there total energy. A Compton scattered photon, with an energy in the 100's of keV range, will then lose far less energy per collision than an electron in the low keV range liberated from the passage of a neutron reaction product through the material. When the average and median energy transfer are plotted as a function of incident electron energy (Figure 4) the difference in the energy loss spectra becomes more apparent. For low energies (up to an incident electron energy of 100 eV) the average and median energy transfer are roughly equal to each other, about half of the incident electron. Past 100 eV average energy increases faster than the median energy transfer implying that

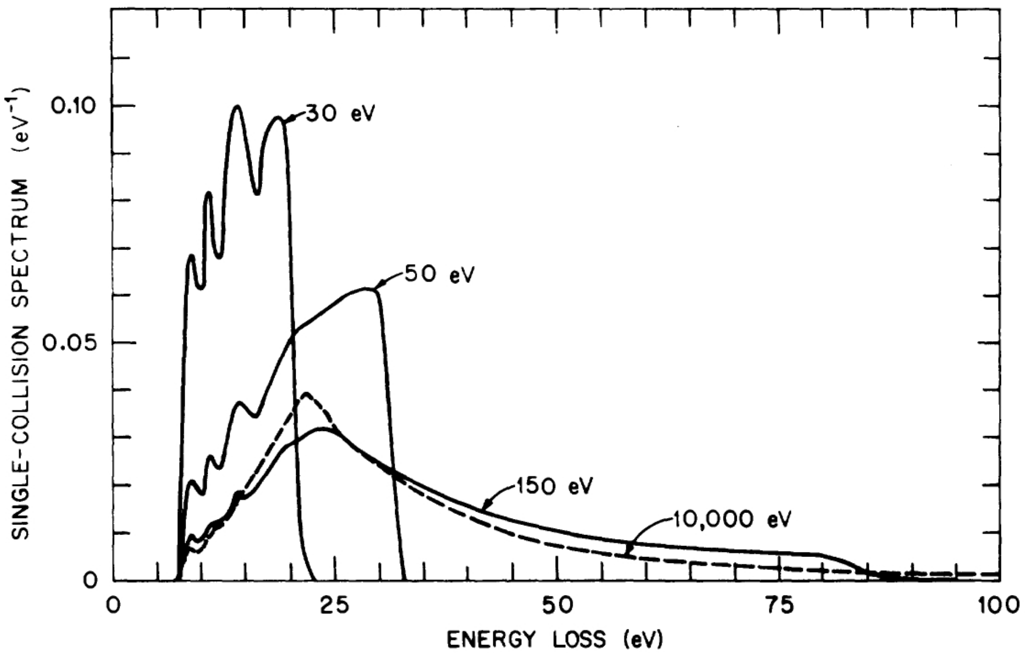


Fig. 3: Single-collision energy loss spectra for electrons in water [1]

while a few collisions result in large energy transfers most of the collisions do not. It is also interesting to note that the average and median do not increase linearly with the incident energy past 100 eV (the ordinate axis is a log scale). In fact, the average energy transferred per collision is mostly bounded by 60 eV even for incident electron energies of $\tilde{1}0$ keV. This is significant because it implies that high energy electrons from photon events will deposit a small fraction of their energy in the material.

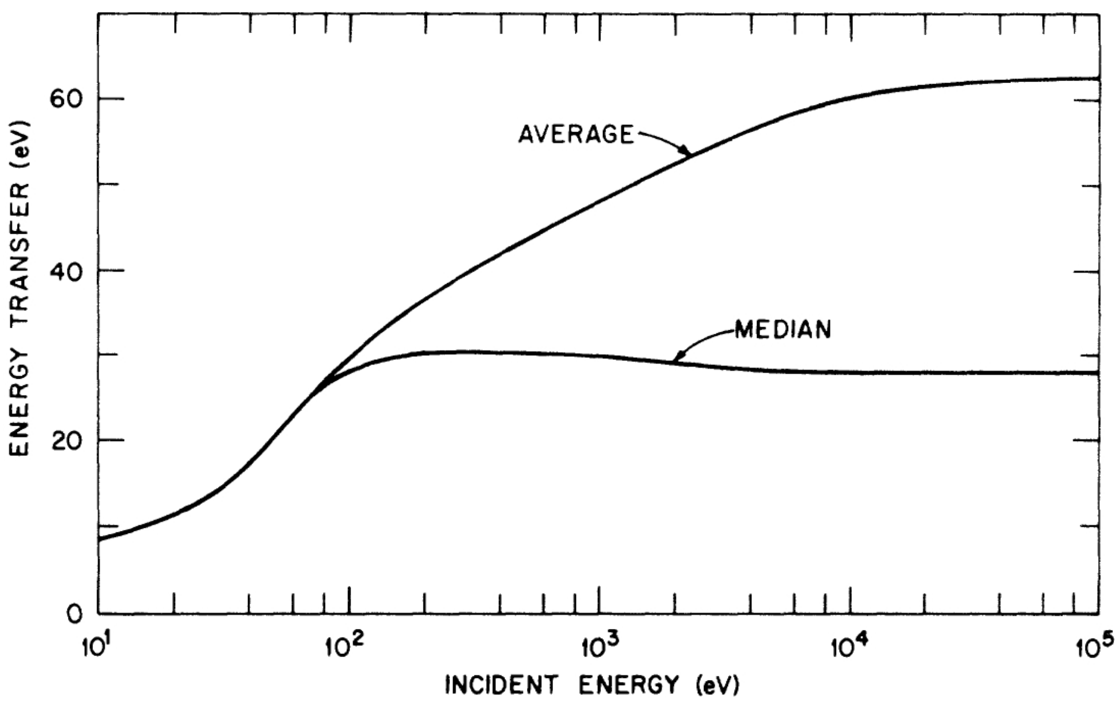


Fig. 4: Average and median energy transfer in liquid water as functions of incident-electron energy [1]

II. Introduction to GEANT4

GEANT4 (GEomentry ANd Tracking) is a free, open source, Monte Carlo based physics simulation toolkit developed and maintained at CERN widely used in the physics community [2], [3], [4]. It is based off of the exsisting FORTRAN based GEANT3, but updated to an object-oriented C++ environment based on an initiative started in 1993. The initiative grew to become an international collaboration of researchers participating in a range of high-energy physics experiments in Europe, Japan, Canada and the United States. As GEANT4 is a toolkit primarily developed for high energy physics, particles are designated according the `PDG` (Particle Data Group) encoding. In addition, the physics processes are referenced according to the standard model. In the standard model particles are divided into two families, bosons (the force carriers such as photons) and fermions (matter). The fermions consist of both hadrons and leptons. Hadrons are particles composed of quarks which are divided into two classes: baryons (three quarks) and mesons (two quarks).h Typical baryons include the neutron and the proton, while an example of a meson is the pion. An example of a lepton is the electron.

A. *Organization of the GEANT4 Toolkit*

The GEANT4 toolkit is divided into eight class categories:

- Run and Event - generation of events and secondary particles.
- Tracking and Track - transport of a particle by analyzing the factors limiting the step size and by applying the relevant physics models.
- Geometry and Magnetic Field - the geometrical definition of a detector (including the computation of the distances to solids) as wells as the management of magnetic fields.
- Particle Definition and Matter - definition of particles and matter.
- Hits and Digitization - the creation of hits and their use for digitization in order to model a detector's readout response.
- Visualization - the visualization of a simulation including the solid geometry, trajectories and hits.
- Interface - the interactions between the toolkit and graphical user interfaces and well as external software.

There are then three classes which must be implemented by the user in order use the toolkit. These classes are:

- `G4VUserDetectorConstruction` which defines the geometry of the simulation,
- `G4VUserPhysicsList` which defines the physics of the simulation, and
- `G4VUserPriamryGeneratorAction` which defines the generation of primary events.

Five additional classes are available for further control over the simulation:

- `G4UserRunAction` which allows for user actions

Expand this section

B. *GEANT4 Tracking and Secondaries*

A GEANT4 simulation starts with a run which contains a set number of events. In GEANT4 the `Run` is the large unit of simulation (represented with a `G4Run` object), which consists of a sequence of events. An event is particular process of interest to the user, such as shooting a single particle at a detector. Typical usage might be to have a run firing 1,000 neutrons at a detector, where each neutron is a single event. Each particle transported in GEANT4 is assigned a unique track ID and a parent ID. The particle that initiates the event is given a parent ID of 0 and a track ID of 1. If the parent particle has a collision, and produces a secondary particle, this secondary particle is then given a parent ID of 1 (corresponding to the first secondary) and a track ID of 2. Secondaries are tracked in GEANT4 utilizing a stack in which the most recent secondary (and its cascade) is tracked first.

Listing 1 provides an example from the verbose output of GEANT4 of the tracking. The initial particle in the event is the neutron because it has a parent ID of 0. The alpha and triton are the secondaries produced by this collision. The alpha is assigned a parent ID of 1 (corresponding to the first generation) with a track ID of 3. The triton is also assigned a parent ID of 1, but with a track ID of 2.

Listing 1: Tracking Example

```
*******************************************************************************************
* G4Track Information:   Particle = neutron,   Track ID = 1,   Parent ID = 0
*******************************************************************************************

Step#    X(mm)    Y(mm)    Z(mm) KinE(MeV)  dE(MeV) StepLeng TrackLeng  NextVolume ProcName
    0        0        0    -6.59   2.5e-08        0        0         0   Absorber initStep
    1        0        0    -3.64         0        0     2.95      2.95   Absorber NeutronInelastic
    :----- List of 2ndaries - #SpawnInStep=  2(Rest= 0,Along= 0,Post= 2), #SpawnTotal=  2 ---------
    :         0        0     -3.64      2.73              triton
    :         0        0     -3.64      2.05               alpha
    :------------------------------------------------------------- EndOf2ndaries Info ---------

*******************************************************************************************
* G4Track Information:   Particle = alpha,   Track ID = 3,   Parent ID = 1
*******************************************************************************************

Step#    X(mm)    Y(mm)    Z(mm) KinE(MeV)  dE(MeV) StepLeng TrackLeng  NextVolume ProcName
    0        0        0    -3.64      2.05        0        0         0   Absorber initStep
    1 -0.000201 0.000128    -3.64      2.01   0.0491 0.000266  0.000266   Absorber ionIoni
    2 -0.00049 0.000312    -3.64      1.93   0.0705 0.000381  0.000647   Absorber ionIoni

*******************************************************************************************
* G4Track Information:   Particle = triton,   Track ID = 2,   Parent ID = 1
*******************************************************************************************

Step#    X(mm)    Y(mm)    Z(mm) KinE(MeV)  dE(MeV) StepLeng TrackLeng  NextVolume ProcName
    0        0        0    -3.64      2.73        0        0         0   Absorber initStep
    1 0.000339 -0.000215    -3.64      2.71   0.0116 0.000447  0.000447   Absorber hIoni
```

III. Methods

A discussion of the steps necessary to implement the simulation of energy deposition in GEANT4 follows. This involved writing the code for the simulation, as well as correctly interpreting the output. As such, this section is organized by first examining the process of setting up the simulation and then will go into the analysis of the results from the toolkit.

A. GEANT4 Implementation

A large focus of this work was on creating a working simulation of the GEANT4 toolkit. Preliminary attempts were made to install GEANT4 on a Windows based machine linking to Microsoft Visual Studio. While these attempts were successful, a larger scale computing environment was desired. GEANT4 was then installed on the University of Tennessee's nuclear engineering computing cluster, along with the necessary visualization drivers and data files. Brief documentation on compiling simple examples on the cluster are available at the necluster wiki [2]. For convenience a subversion repository was created to manage the developed code base, and all source code is available by anonymous checkout from `http://www.murphs-code-repository.googlecode.com/svn/trunk/layeredPolymerTracking`. Revision 360 was the code base used to generate the results shown. The following section provides implementation specific details of the code base used to simulate the energy deposition in thin films. It is organized according to the three base classes that a user must implement in GEANT4, namely `G4VUserDetectorConstruction`, `G4VUserPhysicsList`, and `G4VUserPrimaryGeneratorAction`.

1) Detector Geometry: A detector geometry in GEANT4 is made up of a number of volumes. The largest volume is the `world` volume which contains all other volumes in the detector geometry. Each volume (an instance of `G4VPhysicalVolume`) is created by assigning a position, a pointer to the mother volume and a pointer to its mother volume (or `NULL` if it is the `world` volume). A volume's shape is described by `G4VSolid`, which has a shape and the specific values for each dimension. A volume's full properties is described by a logical volume. A `G4LogicalVolume` includes a pointer to the geometrical properties of the volume (the solid) along with physical characteristics including:

- the material of the volume,
- sensitive detectors of the volume and,
- any magnetic fields.

Listing 17 provides the implementation of the world physical volume. The geometry was set up such that it is possible to define multiple layers of detectors, as shown in Figure 11.

Listing 2: World Physical Volume

```
    // World
```

[2]It should be noted that this example uses the CMAKE build system (as per the GEANT4 recommendation) but a large majority of the examples still use GNUMake for building. This can be accomplished by adding `source /opt/geant4/geant4-9.5p1/share/Geant4-9.5.1/geant4make/geant4make.sh` to the user's `.bashrc`.

```
worldS = new G4Box("World",worldSizeXY, worldSizeXY, worldSizeZ*0.5);
worldLV = new G4LogicalVolume(worldS,defaultMaterial,"World");
worldPV = new G4PVPlacement(0,G4ThreeVector(),worldLV,"World",0,false,0,fCheckOverlaps);
```

The detector was described by creating creating a single layer of neutron absorber and gap material and placing it in another volume (the calorimeter). The containing volume (calorimeter) was placed inside of the the physical world (Listing 18).

Listing 3: Calorimeter Volume

```
// Calorimeter (gap material)
caloS = new G4Tubs("Calorimeter",iRadius,oRadius,caloThickness/2,startAngle,spanAngle);
caloLV = new G4LogicalVolume(caloS,gapMaterial,"Calorimeter");
caloPV = new G4PVPlacement(0,G4ThreeVector(),caloLV,"Calorimeter",worldLV,false,0,
    fCheckOverlaps);
```

The `calorimeter` was the mother volume for each layer. The code was developed such that the simulation of multiple layers can be easily set at compile time or by utilizing a run macro through the `DetectorMessenger` class. Multiple repeated volume can be achieved in GEANT4 through `G4PVReplica` or `G4PVParameterised`. As each of the layers had the same geometry, `G4PVReplica` was chosen as the implementation (Listing 19).

Listing 4: Layer Volume

```
// Layer (Consists of Absorber and Gap)
layerS = new G4Tubs("Layer",iRadius,oRadius,layerThickness/2,startAngle,spanAngle);
layerLV = new G4LogicalVolume(layerS,defaultMaterial,"Layer");
if (nofLayers > 1){
    layerPV = new G4PVReplica("Layer",layerLV,caloLV,kZAxis,nofLayers,layerThickness,-
        caloThickness/2);
}else{
    layerPV = new G4PVPlacement(0,G4ThreeVector(0.0,0.0,0.0),layerLV,"Layer",caloLV,false,0,
        fCheckOverlaps);
}
```

Finally, the neutron absorber and gap material were defined as single cylinders which were then placed in the layer mother volume (Listing 20). The size of these solids (and the materials) could be set either at compile time through `DetectorConstruction` constructor or by using the `DetectorMessenger` in the run macro. Figure 11 shows a rendering of the 10 layers of the detector with the trajectories from a gamma event.

Listing 5: Absorber and Gap Volumes

```
// Absorber
absS = new G4Tubs("Abso",iRadius,oRadius,absThickness/2,startAngle,spanAngle);
absLV = new G4LogicalVolume(absS,absMaterial,"Absorber",0);
absPV = new G4PVPlacement(0,G4ThreeVector(0.0,0.0,-gapThickness/2),absLV,"Absorber",layerLV,
    false,0,fCheckOverlaps);

// Gap
```

```
    gapS = new G4Tubs("Gap",iRadius,oRadius,gapThickness/2,startAngle,spanAngle);
    gapLV = new G4LogicalVolume(gapS,gapMaterial,"Gap",0);
    gapPV = new G4PVPlacement(0,G4ThreeVector(0.0,0.0,absThickness/2),gapLV,"Gap",layerLV,false
        ,0,fCheckOverlaps);
```

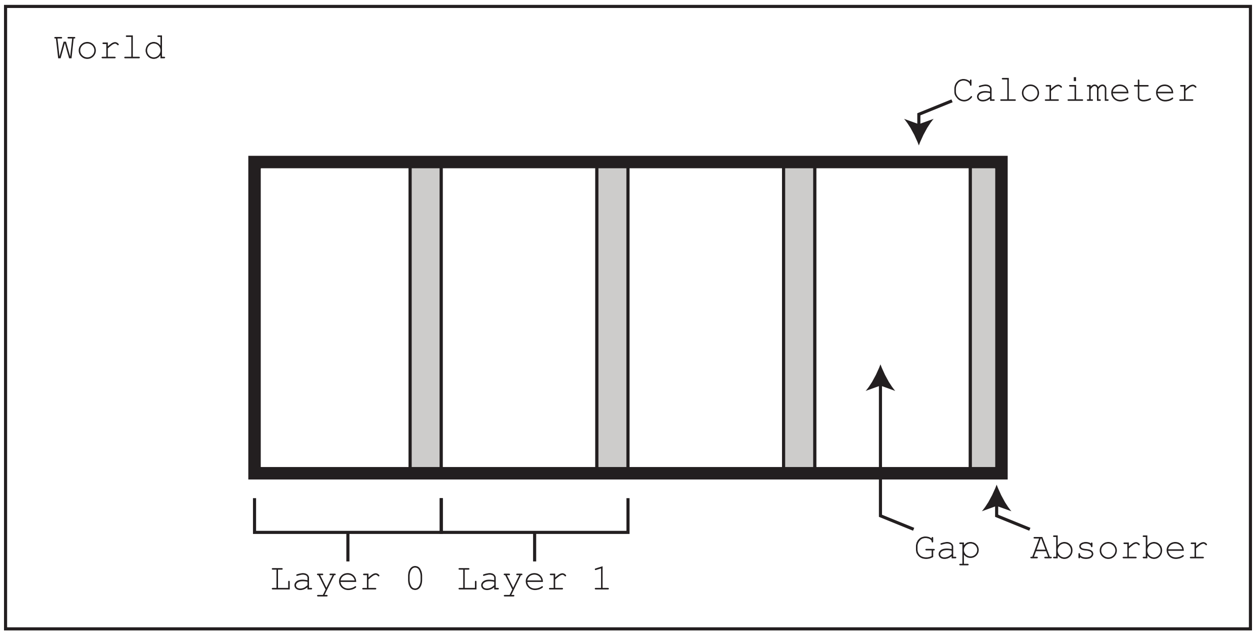


Fig. 5: World, Calorimeter, Layer and Absorber and Gap

2) Physics Lists: The user of the GEANT4 toolkit is responsible for selecting the proper physics processes to model in the `PhysicsList`. This is unlike other transport codes (such as MCNPX) where basic physics are enabled by default and the user only has select the appropriate cards. However, GEANT4 does provide examples of implemented `PhysicsLists` as well as modular physics lists which provide a way to construct a physics list by combing physics list. Thus, extensive use of `G4ModularPhysicsList` was employed to handle the assigning of the physics processes to each particle in the correct order. The physics lists chosen for this simulation are listed below:

- `G4EmStandardPhysics` The electromagnetic physics defines the electrons, muons, and taus along with their corresponding neutrinos. For electrons, the primary concern of this simulation, multiple scattering, electron ionization, and electron bremsstrahlung processes were assigned. In addition the positron is defined and the multiple scattering process, electron ionization process, electron bremsstrahlung process and positron annihilation is assigned [5].
- `G4EmLivermorePhysics` The Livermore physics process extend the `EMStandardPhyiscs` down to low (250 eV) energies. Even lower energies can be reached by including `G4DNAPhysics`. The physics processes extended with `G4EmLivermorePhysics` are the photo-electric effect, Compton scattering, Rayleigh scattering, gamma conversion, Ionisation and Bremsstrahlung[5].

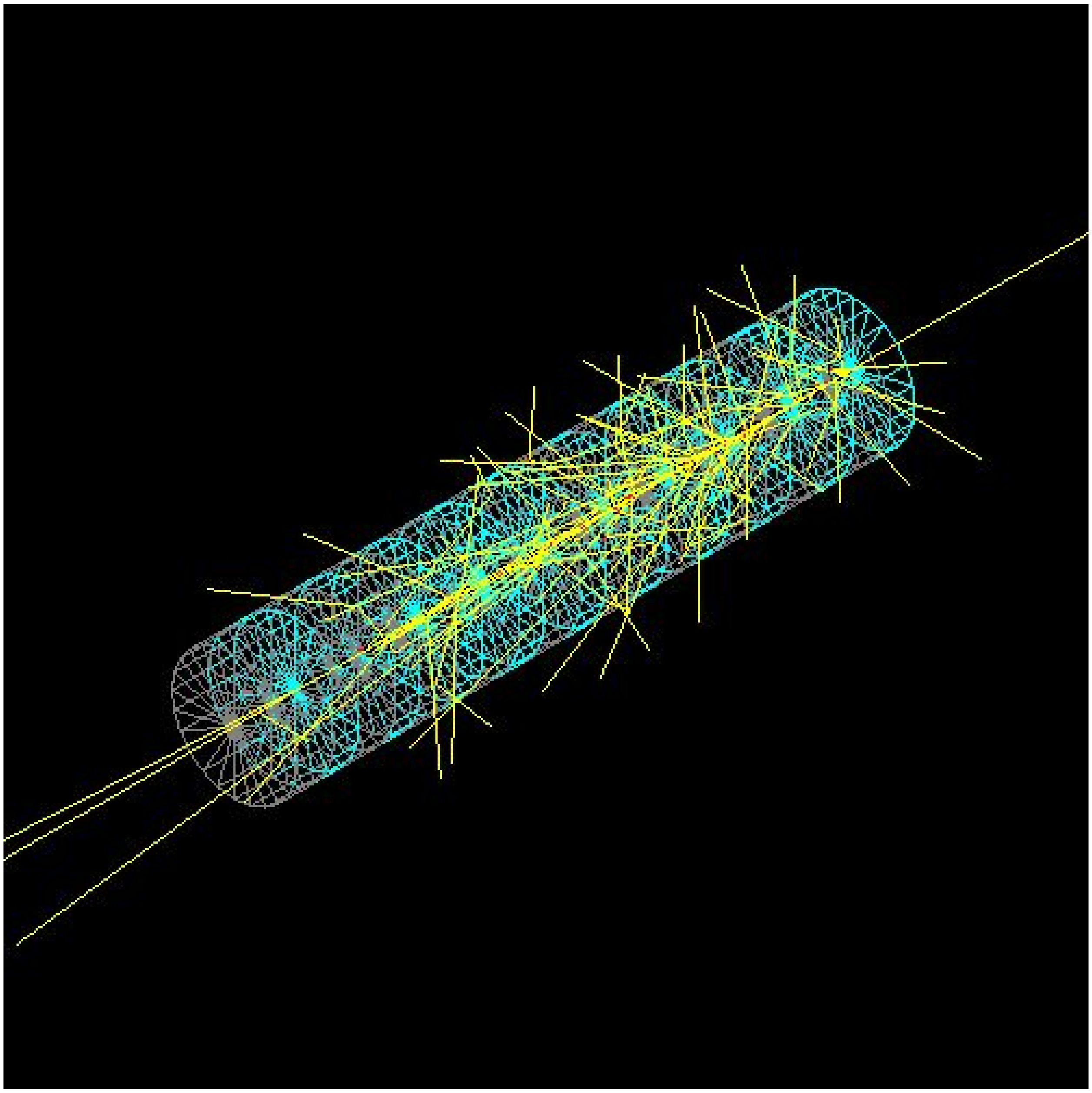


Fig. 6: 10 Layer Detector with a simulated gamma event

- `HadronPhysicsQGSP_BERT_HP` Hadronic physics are included to model the nuclear interactions. The chosen list is a Quark Gluon String Model for energies in the 5-25 GeV range, with a Bertini cascade model until 20 MeV. Once a hadron has an energy of 20 MeV the high precision cross section driven models are applied[6].
- `G4IonPhysics` Finally, to handle the transport of the charged ions resulting from an $^6\mathrm{Li}(\mathrm{n},\alpha)^3\mathrm{H}$ interaction the `G4IonPhysics` list was used.

Listing 6: Implemented Physics List

```
/**
 * PhysicsList
 *
 * Constructs the physics of the simulation
 */
PhysicsList::PhysicsList() : G4VModularPhysicsList() {
    currentDefaultCut   = 10*nm;

    // Adding Physics List
    //RegisterPhysics( new G4EmDNAPhysics());
    RegisterPhysics( new G4EmStandardPhysics());
    RegisterPhysics( new G4EmLivermorePhysics());
    RegisterPhysics( new HadronPhysicsQGSP_BERT_HP());
    RegisterPhysics( new G4IonPhysics());
}
```

Finally, the default cut range was decreased from 1 cm to 1 nm in `SetCuts()` (Listing 22)

Listing 7: Implemented Physics List

```
void PhysicsList::SetCuts(){
    SetDefaultCutValue(10*nm);
}
```

3) Primary Event Generator: The user is responsible for telling the simulation toolkit the primary event to generate. While there is great flexibility to generate any source distribution, a particle gun was chosen for simplicity. `G4ParticleGun` generates primary particle(s) with a given momentum and position without any randomization. The implementation of this is shown in Listing 23.

Listing 8: Primary Event Generator

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

  // default particle kinematic
  G4ParticleDefinition* particleDefinition = G4ParticleTable::GetParticleTable()->FindParticle("e
      -");
  fParticleGun->SetParticlePosition(G4ThreeVector(0.,0.,0.0));
  fParticleGun->SetParticleDefinition(particleDefinition);
  fParticleGun->SetParticleMomentumDirection(G4ThreeVector(0.,0.,1.));
  fParticleGun->SetParticleEnergy(50.*MeV);
}
```

Actual primary particles are generated with `GeneratePrimaries`, which uses the `G4ParticleGun` to determine the vertex of the primary event.

Listing 9: Generate Primaries

```
void PrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{
  // This function is called at the begining of event

  // In order to avoid dependence of PrimaryGeneratorAction
  // on DetectorConstruction class we get world volume
  // from G4LogicalVolumeStore
  G4double worldZHalfLength = 0;
  G4LogicalVolume* worlLV = G4LogicalVolumeStore::GetInstance()->GetVolume("World");
  G4Box* worldBox = 0;
  if ( worlLV) worldBox = dynamic_cast< G4Box*>(worlLV->GetSolid());
  if ( worldBox ) {
    worldZHalfLength = worldBox->GetZHalfLength();
  }
  else  {
    G4cerr << "World volume of box not found." << G4endl;
    G4cerr << "Perhaps you have changed geometry." << G4endl;
    G4cerr << "The gun will be place in the center." << G4endl;
  }

  // Set gun position
  fParticleGun->SetParticlePosition(G4ThreeVector(0., 0., -worldZHalfLength+1*cm));
  fParticleGun->GeneratePrimaryVertex(anEvent);
}
```

B. *Sensitive Detectors and Hits*

GEANT4 offers a myriad of different ways to output the results of a simulation. It is possible to write out every track with the `Verbose = 1` option, create `MultiFunctionalDetector` and `G4VPrimitiveScorer`, or implement a hit and readout based approach [7]. Previous GEANT4 experience included `G4VHit` and `G4VSensitiveDetector`, so this approach was used in this simulation. A hit is defined to be a snapshot of the physical interaction of a track in a sensitive region of a detector. As the user is responsible for implementing `G4VHit` the hit can contain any information about the step, including:

- the position and time of the step,
- the momentum and energy of the track,
- the energy deposition of the step,
- or information about the geometry.

For this simulation any information about the particle that could be recorded was recorded. This included the energy deposition, position of the hit, momentum, kinetic energy, track ID, parent ID, particle definition, volume and copy number (Listing 25).

Listing 10: Calorimeter Hit

```

/**
 * @brief - Hit: a snapshot of the physcial interaction of a track in the sensitive region of a
     detector
 *
 * Contians:
 *   - Particle Information (type and rank (primary, secondary, tertiary ...))
 *   - Positon and time
 *   - momentum and kinetic energy
 *   -  deposition in volume
 *   - geometric information
 */
class CaloHit : public G4VHit {
  public:
    CaloHit(const G4int layer);
    ~CaloHit();

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

  private:
    G4double edep;                /* Energy Deposited at the Hit */
    G4ThreeVector pos;              /* Position of the hit */
    G4double stepLength;            /* Step Length */
    G4ThreeVector momentum;           /* Momentrum of the step */
    G4double kEnergy;                   /* Kinetic Energy of the particle */
    G4int trackID;                /* Track ID */
    G4int parentID;                     /* Parent ID */
        G4ParticleDefinition* particle;     /* Particle Definition */
    G4int particleRank;                 /* Primary, Secondary, etc */
    G4VPhysicalVolume* volume;      /* Physical Volume */
    G4int layerNumber;                  /* Copy Number of Layer */

  public:
    // Setter and Getters
};

typedef G4THitsCollection<CaloHit> CaloHitsCollection;
extern G4Allocator<CaloHit> HitAllocator;

inline void* CaloHit::operator new(size_t){
  void *aHit;
  aHit = (void *) HitAllocator.MallocSingle();
  return aHit;
}

inline void CaloHit::operator delete(void *aHit){
  HitAllocator.FreeSingle((CaloHit*) aHit);
}
```

The G4VSensitiveDetector is attached to a logical volume and is responsible for filling the hit collection. This is accomplished in ProcessHits of CaloSensitiveDetector (Listing 26).

Listing 11: Sensitive Detector

```
/**
 * ProcessHits
 *
 * Adds a hit to the sensitive detector, depending on the step
 */
G4bool CaloSensitiveDetector::ProcessHits(G4Step* aStep,G4TouchableHistory*){

    G4double edep = aStep->GetTotalEnergyDeposit();
    G4double stepLength = aStep->GetStepLength();

    // Getting the copy number
    G4TouchableHistory* touchable = (G4TouchableHistory*)
        (aStep->GetPreStepPoint()->GetTouchable());
    G4int layerIndex = touchable->GetReplicaNumber(1);

    // Creating the hit
    CaloHit* newHit = new CaloHit(layerIndex);
    newHit->SetTrackID(aStep->GetTrack()->GetTrackID());
    newHit->SetParentID(aStep->GetTrack()->GetParentID());
    newHit->SetEdep(edep);
    newHit->SetStepLength(stepLength);
    newHit->SetPosition(aStep->GetPreStepPoint()->GetPosition());
    newHit->SetLayerNumber(layerIndex);
    newHit->SetMomentum(aStep->GetPreStepPoint()->GetMomentum());
    newHit->SetKineticEnergy (aStep->GetPreStepPoint()->GetKineticEnergy());
    newHit->SetParticle(aStep->GetTrack()->GetDefinition());
    newHit->SetVolume(aStep->GetTrack()->GetVolume());

    // Adding the hit to the collection
    hitCollection->insert( newHit );

    return true;
}
```

The simulation was designed so that a separate sensitive detector was assigned to the gap and absorber. While this is not strictly necessary as the geometric position determines what layer of the gap or absorber the hit occurred in, this made the analysis code easier to write. A separate method was written in DetectorConstruction to create the sensitive detectors and assign them to the proper logical volumes (Listing 27) SetSensitiveDetectors() is called from the the constructor of DetectorConstruction.

Listing 12: Creating Sensitive Detectors

```
/**
 * SetSensitiveDetectors
 *
 * Setting the Sensitive Detectors of the Detector
 */
void DetectorConstruction::SetSensitiveDetectors(){
    G4SDManager* SDman = G4SDManager::GetSDMpointer();
    absSD = new CaloSensitiveDetector("SD/AbsSD","AbsHitCollection");
    SDman->AddNewDetector(absSD);
    absLV->SetSensitiveDetector(absSD);

    gapSD = new CaloSensitiveDetector("SD/GapSD","GapHitCollection");
    SDman->AddNewDetector(gapSD);
    gapLV->SetSensitiveDetector(gapSD);
}
```

C. *Analysis*

Analysis of hit collection was preformed with ROOT. Once again there are other options (notably OpenScientist) but previous experience was why ROOT was selected as the base for the Analysis framework. A singleton class was written for the analysis which processed the hit collections, assigning the various results to root histograms. User action classes `EventAction` and `RunAction` are called at the beginning and end of each run and event, respectively (Listing 28,29). These classes allowed for the analysis code to be independent of the simulation.

Listing 13: Event Action

```
EventAction::EventAction() : G4UserEventAction(){
    // Nothing to be Done Here
}

/**
 * BeginOfEventAction
 *
 * @param const G4Event* event - event to be processed
 *
 * At the begining of an event we want to clear all the event
 * accumulation variables.
 */
void EventAction::BeginOfEventAction(const G4Event* event){
    Analysis::GetInstance()->PrepareNewEvent(event);
}

/**
 * EndOfEventAction
 *
 * @param const G4Event* event - event to be processed
 *
 * At the end of an event we want to call analysis to proccess
 * this event, and record the useful information.
 */
void EventAction::EndOfEventAction(const G4Event* event){
    Analysis::GetInstance()->EndOfEvent(event);
}
```

Listing 14: Run Action

```
RunAction::RunAction() : G4UserRunAction(){ }

void RunAction::BeginOfRunAction(const G4Run* run){
  G4cout<<"Starting  run: " << run->GetRunID()<< G4endl;
    Analysis::GetInstance()->PrepareNewRun(run);
}

void RunAction::EndOfRunAction(const G4Run* aRun){
    Analysis::GetInstance()->EndOfRun(aRun);
}
```

D. Determination of Energy Deposition

The energy deposition of an event is calculated by the sum of all of the energy deposited by individual hits in the sensitive detector (Equation 5). While it is possible to break down the energy deposition by which physics process caused the deposition, this was not implemented in order to avoid over complication.

$$E_{\text{dep,event}} = \sum E_{\text{dep,hit}} \tag{1}$$

`ProcessHitCollection` is called at the end of each event (Listing 30). Each hit is accessed and the layer at which it occurs is determined[3]. In addition the name of the volume is determined, and the energy deposition of the hit is added to the energy deposition of the event. If the hit occurred in the `absorber` layer and the particle is an electron the kinetic energy of that hit is also recorded.

Listing 15: Process Hit Collection

```
/**
 * ProcessHitCollection
 *
 * @param G4VHitsCollection *hc
 */
void Analysis::ProcessHitCollection(G4VHitsCollection *hc,G4int eventID){

    // Looping through the hit collection
```

[3]C arrays start at 0, so memory is allocated for one more than the total number of layers. This allows for `NUMLAYERS+1` to be used an index into the histogram for the total of all layers in the material (either `gap` or `absorber`).

```
    G4double hitColEDepTot_Abs[NUMLAYERS+1];   // Total EDep (abs) for Hit Collection
    G4double hitColEDepTot_Gap[NUMLAYERS+1];   // Total EDep (gap) for Hit Collection
    G4int PID;                                 // Parent ID
    for(int i= 0; i < NUMLAYERS+1; i++){
        hitColEDepTot_Abs[i] = 0.0;
        hitColEDepTot_Gap[i] = 0.0;
    }

    // Energy Deposition of the event
    for(G4int i = 0; i < hc->GetSize(); i++){
        CaloHit* hit = (CaloHit*) hc->GetHit(i);

        G4double eDep = hit->GetEdep();
        G4int layerNum = hit->GetLayerNumber();
        if (strcmp(hit->GetVolume()->GetName(),"Gap")){
            // Hit occured in the Gap
            hitColEDepTot_Gap[layerNum] += eDep;
            (hHitTotEDepGap[layerNum])->Fill(eDep);
        }else if(strcmp(hit->GetVolume()->GetName(),"Absorber")){
            // Hit occured in the Abs
            hitColEDepTot_Abs[layerNum] += eDep;
            (hHitTotEDepAbs[layerNum])->Fill(eDep);

            /* Is this a secondary electron of the event? */
            if(hit->GetParticle()->GetPDGEncoding() == 11){
                PID = hit->GetParentID();
                if (PID < NUMPID){
                    (hSecElecKinAbs[layerNum][PID])->Fill(hit->GetKineticEnergy());
                }
            }
        }
        else{
            G4cout<<"ERROR - Unkown Volume for sensitive detector"<<G4endl;
        }
    }

    // Adding this Hit collection's energy deposited to event total
    for (int i = 0; i < NUMLAYERS; i++){
        // Incrementing each individual bin
        eventEDepTot_Abs[i] += hitColEDepTot_Abs[i];
        eventEDepTot_Gap[i] += hitColEDepTot_Gap[i];

        // Last bin is Calorimter Total (all Abs layers and all Gap layers)
        eventEDepTot_Abs[NUMLAYERS] += hitColEDepTot_Abs[i];
        eventEDepTot_Gap[NUMLAYERS] += hitColEDepTot_Gap[i];
    }
}
```

Finally, a run macro was written to control the entire run (Listing 31). The material and thickness of the detector are declared (made possible by the use of `DetectorMessenger`), and then the detector is dynamically updated. A ^{60}Co source is simulated by shooting photons of the 1.1732 MeV and 1.3325 MeV. The source particle is then changed to a neutron, and thermal (0.025 eV) neutrons are shot at the detector. The thickness of the absorber is then increased, the geometry updated, and the entire process repeated. As these runs tend to take a large amount of time, GEANT4 was parallelized for use with MPI to take advantage of the cluster computing power.

Listing 16: Run Macro

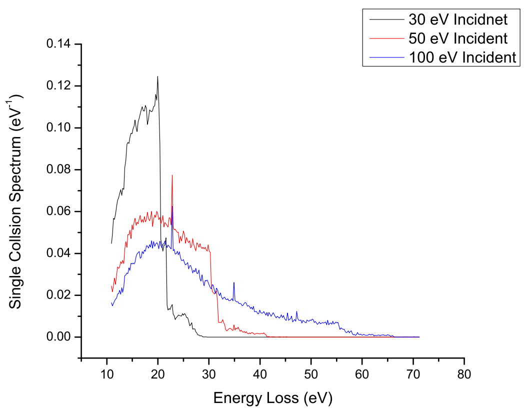
```
#
/tracking/verbose 0
#
# Setting up the detector
#
/PolymerTransport/det/setAbsMat PS_Detector
/PolymerTransport/det/setGapMat G4_POLYSTYRENE
/PolymerTransport/det/setGapThick 0.3175 cm
#
/PolymerTransport/det/setAbsThick 15 um
/PolymerTransport/det/update
# Cobalt 60
/gun/particle gamma
/gun/direction 0 0 1
/gun/energy 1.1732 MeV
/run/beamOn 500000000 # 500 Million
/gun/energy 1.3325 MeV
/run/beamOn 500000000 # 500 Million
# Neutron
/gun/particle neutron
/gun/energy 0.025 eV
/run/beamOn 1000000 # 1 Million
#
/PolymerTransport/det/setAbsThick 25 um
/PolymerTransport/det/update
```

IV. Simulation Validation

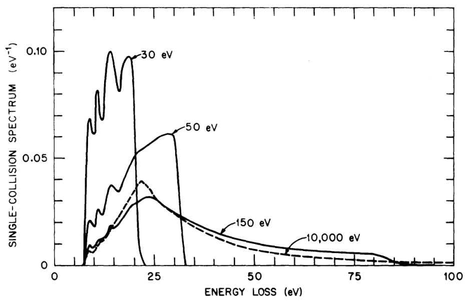
GEANT4 is a toolkit implemented by the user so extensive efforts were completed in order to validate the results and ensure no bugs exists. First steps were taken (for small runs) to compute the energy deposition for small runs by hand in order to make sure they agreeded with the analysis code. In addition the reaction products of the $^{6}\text{Li}(\text{n},\alpha)^{3}\text{H}$ were checked to make sure that they agreeded with the published values [4]. The GEANT4 simulation was validated by comparing the single collision energy loss spectra in water and by comparing the simulation energy deposition to that of a measured spectra.

A. Energy Deposition Validation

The energy deposition was tested by reproducing the single collision energy loss spectra in water[5]. The `PhysicsList` was extended to include `G4DNAPhysics` and the detector material was set to the NIST definition contained in the toolkit with `G4Material* H20 = man->FindOrBuildMaterial("G4_WATER")`. In general there was excellent agreement between the simulated energy spectra and a previously published spectra[1]. The simulated spectra had much better resolution at fine energies (corresponding to discrete states) of which Turners did not.



(a) Simulated



(b) Single-collision energy loss spectra for electrons in water [1]

(c) Published

Fig. 7: Single Collision Energy Loss of Water

[4]GEANT4 4.9.2.p01 contains an error in which extra photons are generated, This has been fixed in the release used, 4.9.5p1

[5]An analysis class was not written for this simulation. Instead the verbosity of the simulation was set to `verbose=1` in the run macro. The first ionisation collision (`e-_G4DNAIonisation`) was then extracted with `sed -n '/ParentID = 0/,/e-_G4DNAIonisation/p' G4OutputFileName.txt| grep "e-\_G4DNAIonisatioin" | awk '${print $5}' '`

B. Spectra Validation

The simulated energy deposition is not the directly equivilant to light collected on the PMT because the scintillation process and light collection is not modeled. However, it is well known that scintillation follows the energy deposition[8]. Thus, up to scaling contants, the energy deposition can be considered equivilant to the scintillation and representative of the measured spectra. Rather than attempting to back out these scaling contants the weighted average of spectra were used in which integration and normalization removes these fudge factors. The simulation was validated by computing the weighted average of the energy deposition 2 and comparing it to the spectra average defined in 3. There is excellent agreement between the measured gamma weighted average (right ordinate axis) and the average energy deposition from a ^{60}Co source (left ordinate axis). Non-linearity is observed for films less than 200 μm, this is evidance that the cascade electrons from the Compton electron are eneregetic enough that the range of the electrons is much greater than the thickness of the film and leave the film without colliding to an energy in which the energy deposition is linear (Figure 4).

$$< E >= \frac{\int_0^\infty \phi(E) E dE}{\int_0^\infty \phi(E) dE} \text{where} \tag{2}$$

$$< \mu >= \frac{\int_0^\infty f(x) x dx}{\int_0^\infty x(x) dx} \text{where} \tag{3}$$

The comparison between the average energy deposition and measured channel allows for the a relationship to be drawn between the energy deposited and the channel number. This is completed by an taking an average of the ratio between the average channel number (Equation 3 and the average energy deposition (Equation 2). This ratio is defined in Equation 4. This quantity is defined seperately for neturons and gammas.

$$\eta = \sum_t \frac{< E >}{< \mu} \tag{4}$$

Fig. 8: Gamma Simulation Agreement

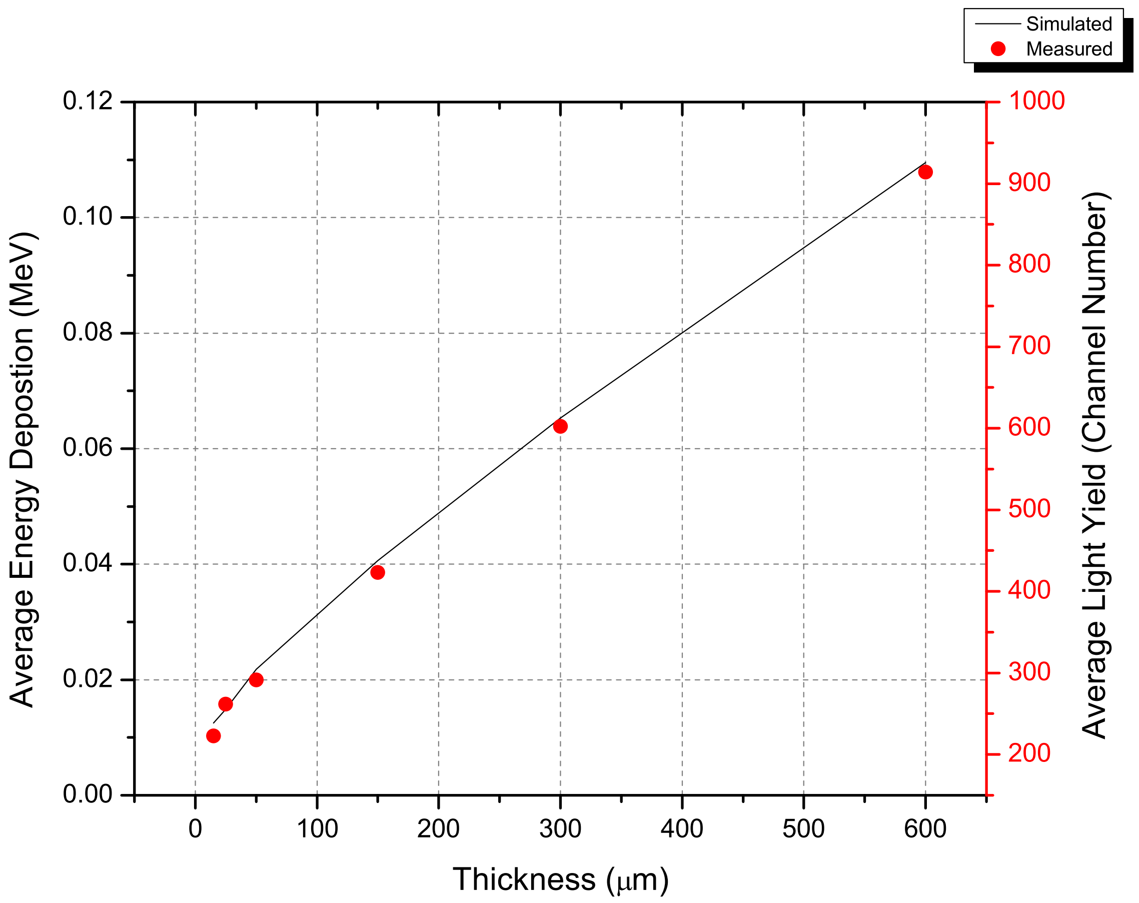
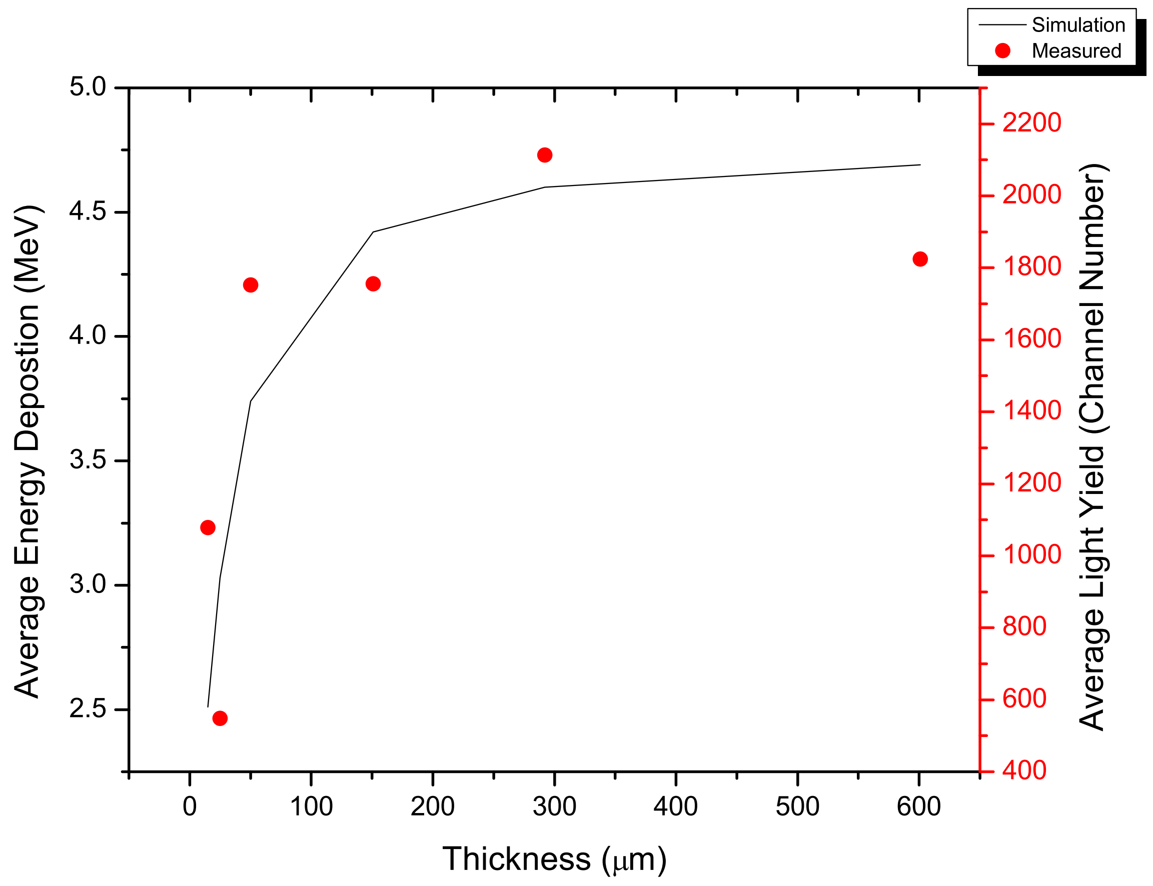


Fig. 9: Neutron Simulation Agreement



V. Methods

A discussion of the steps necessary to implement the simulation of energy deposition in GEANT4 follows. This involved writing the code for the simulation, as well as correctly interpreting the output. As such, this section is organized by first examining the process of setting up the simulation and then will go into the analysis of the results from the toolkit.

A. GEANT4 Implementation

A large focus of this work was on creating a working simulation of the GEANT4 toolkit. Preliminary attempts were made to install GEANT4 on a Windows based machine linking to Microsoft Visual Studio. While these attempts were successful, a larger scale computing environment was desired. GEANT4 was then installed on the University of Tennessee's nuclear engineering computing cluster, along with the necessary visualization drivers and data files. Brief documentation on compiling simple examples on the cluster are available at the necluster wiki [6]. For convenience a subversion repository was created to manage the developed code base, and all source code is available by anonymous checkout from `http://www.murphs-code-repository.googlecode.com/svn/trunk/layeredPolymerTracking`. Revision 360 was the code base used to generate the results shown. The following section provides implementation specific details of the code base used to simulate the energy deposition in thin films. It is organized according to the three base classes that a user must implement in GEANT4, namely `G4VUserDetectorConstruction`, `G4VUserPhysicsList`, and `G4VUserPrimaryGeneratorAction`.

1) Detector Geometry: A detector geometry in GEANT4 is made up of a number of volumes. The largest volume is the `world` volume which contains all other volumes in the detector geometry. Each volume (an instance of `G4VPhysicalVolume`) is created by assigning a position, a pointer to the mother volume and a pointer to its mother volume (or `NULL` if it is the `world` volume). A volume's shape is described by `G4VSolid`, which has a shape and the specific values for each dimension. A volume's full properties is described by a logical volume. A `G4LogicalVolume` includes a pointer to the geometrical properties of the volume (the solid) along with physical characteristics including:

- the material of the volume,
- sensitive detectors of the volume and,
- any magnetic fields.

Listing 17 provides the implementation of the world physical volume. The geometry was set up such that it is possible to define multiple layers of detectors, as shown in Figure 11.

Listing 17: World Physical Volume

```
   // World
```

[6]It should be noted that this example uses the CMAKE build system (as per the GEANT4 recommendation) but a large majority of the examples still use GNUMake for building. This can be accomplished by adding `source /opt/geant4/geant4-9.5p1/share/Geant4-9.5.1/geant4make/geant4make.sh` to the user's `.bashrc`.

```
    worldS = new G4Box("World",worldSizeXY, worldSizeXY, worldSizeZ*0.5);
    worldLV = new G4LogicalVolume(worldS,defaultMaterial,"World");
    worldPV = new G4PVPlacement(0,G4ThreeVector(),worldLV,"World",0,false,0,fCheckOverlaps);
```

The detector was described by creating creating a single layer of neutron absorber and gap material and placing it in another volume (the calorimeter). The containing volume (calorimeter) was placed inside of the the physical world (Listing 18).

Listing 18: Calorimeter Volume

```
    // Calorimeter (gap material)
    caloS = new G4Tubs("Calorimeter",iRadius,oRadius,caloThickness/2,startAngle,spanAngle);
    caloLV = new G4LogicalVolume(caloS,gapMaterial,"Calorimeter");
    caloPV = new G4PVPlacement(0,G4ThreeVector(),caloLV,"Calorimeter",worldLV,false,0,
        fCheckOverlaps);
```

The `calorimeter` was the mother volume for each layer. The code was developed such that the simulation of multiple layers can be easily set at compile time or by utilizing a run macro through the `DetectorMessenger` class. Multiple repeated volume can be achieved in GEANT4 through `G4PVReplica` or `G4PVParameterised`. As each of the layers had the same geometry, `G4PVReplica` was chosen as the implementation (Listing 19).

Listing 19: Layer Volume

```
    // Layer (Consists of Absorber and Gap)
    layerS = new G4Tubs("Layer",iRadius,oRadius,layerThickness/2,startAngle,spanAngle);
    layerLV = new G4LogicalVolume(layerS,defaultMaterial,"Layer");
    if (nofLayers > 1){
        layerPV = new G4PVReplica("Layer",layerLV,caloLV,kZAxis,nofLayers,layerThickness,-
            caloThickness/2);
    }else{
        layerPV = new G4PVPlacement(0,G4ThreeVector(0.0,0.0,0.0),layerLV,"Layer",caloLV,false,0,
            fCheckOverlaps);
    }
```

Finally, the neutron absorber and gap material were defined as single cylinders which were then placed in the layer mother volume (Listing 20). The size of these solids (and the materials) could be set either at compile time through `DetectorConstruction` constructor or by using the `DetectorMessenger` in the run macro. Figure 11 shows a rendering of the 10 layers of the detector with the trajectories from a gamma event.

Listing 20: Absorber and Gap Volumes

```
    // Absorber
    absS = new G4Tubs("Abso",iRadius,oRadius,absThickness/2,startAngle,spanAngle);
    absLV = new G4LogicalVolume(absS,absMaterial,"Absorber",0);
    absPV = new G4PVPlacement(0,G4ThreeVector(0.0,0.0,-gapThickness/2),absLV,"Absorber",layerLV,
        false,0,fCheckOverlaps);

    // Gap
    gapS = new G4Tubs("Gap",iRadius,oRadius,gapThickness/2,startAngle,spanAngle);
    gapLV = new G4LogicalVolume(gapS,gapMaterial,"Gap",0);
    gapPV = new G4PVPlacement(0,G4ThreeVector(0.0,0.0,absThickness/2),gapLV,"Gap",layerLV,false
        ,0,fCheckOverlaps);
```

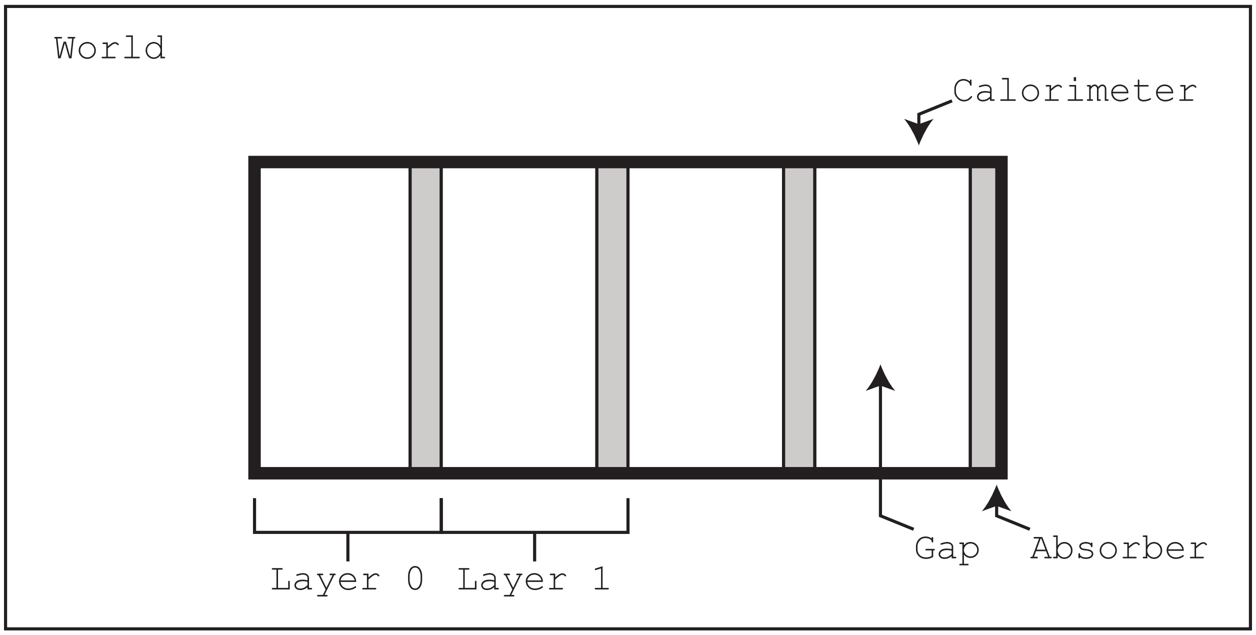


Fig. 10: World, Calorimeter, Layer and Absorber and Gap

2) Physics Lists: The user of the GEANT4 toolkit is responsible for selecting the proper physics processes to model in the `PhysicsList`. This is unlike other transport codes (such as MCNPX) where basic physics are enabled by default and the user only has select the appropriate cards. However, GEANT4 does provide examples of implemented `PhysicsLists` as well as modular physics lists which provide a way to construct a physics list by combing physics list. Thus, extensive use of `G4ModularPhysicsList` was employed to handle the assigning of the physics processes to each particle in the correct order. The physics lists chosen for this simulation are listed below:

- `G4EmStandardPhysics` The electromagnetic physics defines the electrons, muons, and taus along with their corresponding neutrinos. For electrons, the primary concern of this simulation, multiple scattering, electron ionization, and electron bremsstrahlung processes were assigned. In addition the positron is defined and the multiple scattering process, electron ionization process, electron bremsstrahlung process and positron annihilation is assigned [5].
- `G4EmLivermorePhysics` The Livermore physics process extend the `EMStandardPhyiscs` down to low (250 eV) energies. Even lower energies can be reached by including `G4DNAPhysics`. The physics processes extended with `G4EmLivermorePhysics` are the photo-electric effect, Compton scattering, Rayleigh scattering, gamma conversion, Ionisation and Bremsstrahlung[5].

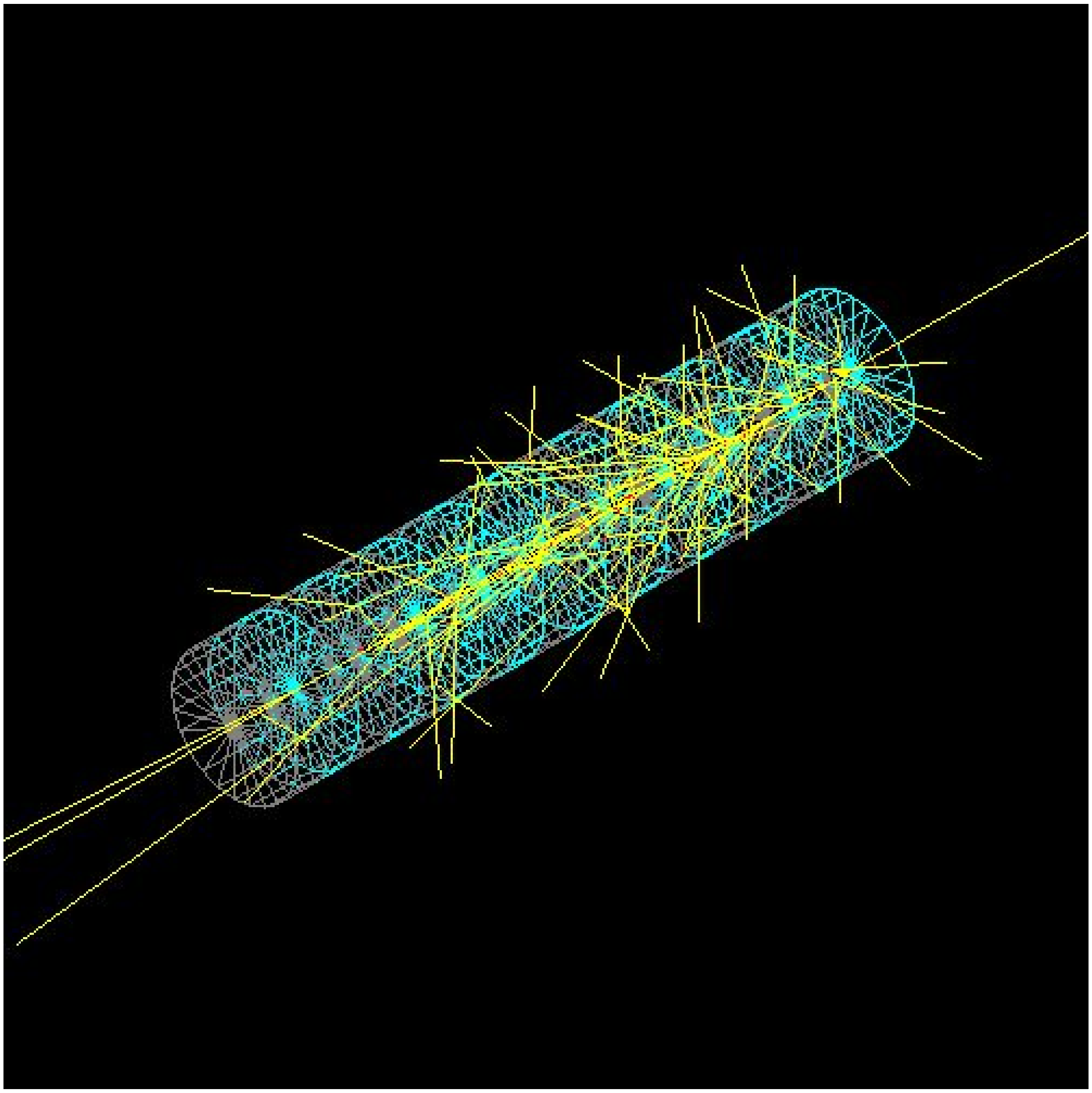


Fig. 11: 10 Layer Detector with a simulated gamma event

- `HadronPhysicsQGSP_BERT_HP` Hadronic physics are included to model the nuclear interactions. The chosen list is a Quark Gluon String Model for energies in the 5-25 GeV range, with a Bertini cascade model until 20 MeV. Once a hadron has an energy of 20 MeV the high precision cross section driven models are applied[6].
- `G4IonPhysics` Finally, to handle the transport of the charged ions resulting from an $^6\mathrm{Li}(\mathrm{n}, \alpha)^3\mathrm{H}$ interaction the `G4IonPhysics` list was used.

Listing 21: Implemented Physics List

```
/**
 * PhysicsList
 *
 * Constructs the physics of the simulation
 */
PhysicsList::PhysicsList() : G4VModularPhysicsList() {
    currentDefaultCut   = 10*nm;

    // Adding Physics List
    //RegisterPhysics( new G4EmDNAPhysics());
    RegisterPhysics( new G4EmStandardPhysics());
    RegisterPhysics( new G4EmLivermorePhysics());
    RegisterPhysics( new HadronPhysicsQGSP_BERT_HP());
    RegisterPhysics( new G4IonPhysics());
}
```

Finally, the default cut range was decreased from 1 cm to 1 nm in `SetCuts()` (Listing 22)

Listing 22: Implemented Physics List

```
void PhysicsList::SetCuts(){
    SetDefaultCutValue(10*nm);
}
```

3) Primary Event Generator: The user is responsible for telling the simulation toolkit the primary event to generate. While there is great flexibility to generate any source distribution, a particle gun was chosen for simplicity. `G4ParticleGun` generates primary particle(s) with a given momentum and position without any randomization. The implementation of this is shown in Listing 23.

Listing 23: Primary Event Generator

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

  // default particle kinematic
  G4ParticleDefinition* particleDefinition = G4ParticleTable::GetParticleTable()->FindParticle("e
      -");
  fParticleGun->SetParticlePosition(G4ThreeVector(0.,0.,0.0));
  fParticleGun->SetParticleDefinition(particleDefinition);
  fParticleGun->SetParticleMomentumDirection(G4ThreeVector(0.,0.,1.));
  fParticleGun->SetParticleEnergy(50.*MeV);
}
```

Actual primary particles are generated with `GeneratePrimaries`, which uses the `G4ParticleGun` to determine the vertex of the primary event.

Listing 24: Generate Primaries

```
void PrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{
  // This function is called at the begining of event

  // In order to avoid dependence of PrimaryGeneratorAction
  // on DetectorConstruction class we get world volume
  // from G4LogicalVolumeStore
  G4double worldZHalfLength = 0;
  G4LogicalVolume* worlLV = G4LogicalVolumeStore::GetInstance()->GetVolume("World");
  G4Box* worldBox = 0;
  if ( worlLV) worldBox = dynamic_cast< G4Box*>(worlLV->GetSolid());
  if ( worldBox ) {
    worldZHalfLength = worldBox->GetZHalfLength();
  }
  else  {
    G4cerr << "World volume of box not found." << G4endl;
    G4cerr << "Perhaps you have changed geometry." << G4endl;
    G4cerr << "The gun will be place in the center." << G4endl;
  }

  // Set gun position
  fParticleGun->SetParticlePosition(G4ThreeVector(0., 0., -worldZHalfLength+1*cm));
  fParticleGun->GeneratePrimaryVertex(anEvent);
}
```

B. *Sensitive Detectors and Hits*

GEANT4 offers a myriad of different ways to output the results of a simulation. It is possible to write out every track with the `Verbose = 1` option, create `MultiFunctionalDetector` and `G4VPrimitiveScorer`, or implement a hit and readout based approach [7]. Previous GEANT4 experience included `G4VHit` and `G4VSensitiveDetector`, so this approach was used in this simulation. A hit is defined to be a snapshot of the physical interaction of a track in a sensitive region of a detector. As the user is responsible for implementing `G4VHit` the hit can contain any information about the step, including:

- the position and time of the step,
- the momentum and energy of the track,
- the energy deposition of the step,
- or information about the geometry.

For this simulation any information about the particle that could be recorded was recorded. This included the energy deposition, position of the hit, momentum, kinetic energy, track ID, parent ID, particle definition, volume and copy number (Listing 25).

Listing 25: Calorimeter Hit

```

/**
 * @brief - Hit: a snapshot of the physcial interaction of a track in the sensitive region of a
     detector
 *
 * Contians:
 *  - Particle Information (type and rank (primary, secondary, tertiary ...))
 *  - Positon and time
 *  - momentum and kinetic energy
 *  -  deposition in volume
 *  - geometric information
 */
class CaloHit : public G4VHit {
  public:
    CaloHit(const G4int layer);
    ~CaloHit();

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

  private:
    G4double edep;                /* Energy Deposited at the Hit */
    G4ThreeVector pos;              /* Position of the hit */
    G4double stepLength;            /* Step Length */
    G4ThreeVector momentum;           /* Momentrum of the step */
    G4double kEnergy;                   /* Kinetic Energy of the particle */
    G4int trackID;                /* Track ID */
    G4int parentID;                     /* Parent ID */
        G4ParticleDefinition* particle;     /* Particle Definition */
    G4int particleRank;                 /* Primary, Secondary, etc */
    G4VPhysicalVolume* volume;      /* Physical Volume */
    G4int layerNumber;                  /* Copy Number of Layer */

  public:
    // Setter and Getters
};

typedef G4THitsCollection<CaloHit> CaloHitsCollection;
extern G4Allocator<CaloHit> HitAllocator;

inline void* CaloHit::operator new(size_t){
  void *aHit;
  aHit = (void *) HitAllocator.MallocSingle();
  return aHit;
}

inline void CaloHit::operator delete(void *aHit){
  HitAllocator.FreeSingle((CaloHit*) aHit);
}
```

The `G4VSensitiveDetector` is attached to a logical volume and is responsible for filling the hit collection. This is accomplished in `ProcessHits` of `CaloSensitiveDetector` (Listing 26).

Listing 26: Sensitive Detector

```
/**
 * ProcessHits
 *
 * Adds a hit to the sensitive detector, depending on the step
 */
G4bool CaloSensitiveDetector::ProcessHits(G4Step* aStep,G4TouchableHistory*){

    G4double edep = aStep->GetTotalEnergyDeposit();
    G4double stepLength = aStep->GetStepLength();

    // Getting the copy number
    G4TouchableHistory* touchable = (G4TouchableHistory*)
        (aStep->GetPreStepPoint()->GetTouchable());
    G4int layerIndex = touchable->GetReplicaNumber(1);

    // Creating the hit
    CaloHit* newHit = new CaloHit(layerIndex);
    newHit->SetTrackID(aStep->GetTrack()->GetTrackID());
    newHit->SetParentID(aStep->GetTrack()->GetParentID());
    newHit->SetEdep(edep);
    newHit->SetStepLength(stepLength);
    newHit->SetPosition(aStep->GetPreStepPoint()->GetPosition());
    newHit->SetLayerNumber(layerIndex);
    newHit->SetMomentum(aStep->GetPreStepPoint()->GetMomentum());
    newHit->SetKineticEnergy (aStep->GetPreStepPoint()->GetKineticEnergy());
    newHit->SetParticle(aStep->GetTrack()->GetDefinition());
    newHit->SetVolume(aStep->GetTrack()->GetVolume());

    // Adding the hit to the collection
    hitCollection->insert( newHit );

    return true;
}
```

The simulation was designed so that a separate sensitive detector was assigned to the gap and absorber. While this is not strictly necessary as the geometric position determines what layer of the gap or absorber the hit occurred in, this made the analysis code easier to write. A separate method was written in `DetectorConstruction` to create the sensitive detectors and assign them to the proper logical volumes (Listing 27) `SetSensitiveDetectors()` is called from the the constructor of `DetectorConstruction`.

Listing 27: Creating Sensitive Detectors

```
/**
 * SetSensitiveDetectors
 *
 * Setting the Sensitive Detectors of the Detector
 */
void DetectorConstruction::SetSensitiveDetectors(){
    G4SDManager* SDman = G4SDManager::GetSDMpointer();
    absSD = new CaloSensitiveDetector("SD/AbsSD","AbsHitCollection");
    SDman->AddNewDetector(absSD);
    absLV->SetSensitiveDetector(absSD);

    gapSD = new CaloSensitiveDetector("SD/GapSD","GapHitCollection");
    SDman->AddNewDetector(gapSD);
    gapLV->SetSensitiveDetector(gapSD);
}
```

C. Analysis

Analysis of hit collection was preformed with ROOT. Once again there are other options (notably OpenScientist) but previous experience was why ROOT was selected as the base for the Analysis framework. A singleton class was written for the analysis which processed the hit collections, assigning the various results to root histograms. User action classes `EventAction` and `RunAction` are called at the beginning and end of each run and event, respectively (Listing 28,29). These classes allowed for the analysis code to be independent of the simulation.

Listing 28: Event Action

```
EventAction::EventAction() : G4UserEventAction(){
    // Nothing to be Done Here
}

/**
 * BeginOfEventAction
 *
 * @param const G4Event* event - event to be processed
 *
 * At the begining of an event we want to clear all the event
 * accumulation variables.
 */
void EventAction::BeginOfEventAction(const G4Event* event){
    Analysis::GetInstance()->PrepareNewEvent(event);
}

/**
 * EndOfEventAction
 *
 * @param const G4Event* event - event to be processed
 *
 * At the end of an event we want to call analysis to proccess
 * this event, and record the useful information.
 */
void EventAction::EndOfEventAction(const G4Event* event){
    Analysis::GetInstance()->EndOfEvent(event);
}
```

Listing 29: Run Action

```
RunAction::RunAction() : G4UserRunAction(){ }

void RunAction::BeginOfRunAction(const G4Run* run){
  G4cout<<"Starting  run: " << run->GetRunID()<< G4endl;
    Analysis::GetInstance()->PrepareNewRun(run);
}

void RunAction::EndOfRunAction(const G4Run* aRun){
    Analysis::GetInstance()->EndOfRun(aRun);
}
```

D. Determination of Energy Deposition

The energy deposition of an event is calculated by the sum of all of the energy deposited by individual hits in the sensitive detector (Equation 5). While it is possible to break down the energy deposition by which physics process caused the deposition, this was not implemented in order to avoid over complication.

$$E_{\text{dep,event}} = \sum E_{\text{dep,hit}} \tag{5}$$

`ProcessHitCollection` is called at the end of each event (Listing 30). Each hit is accessed and the layer at which it occurs is determined[7]. In addition the name of the volume is determined, and the energy deposition of the hit is added to the energy deposition of the event. If the hit occurred in the `absorber` layer and the particle is an electron the kinetic energy of that hit is also recorded.

Listing 30: Process Hit Collection

```
/**
 * ProcessHitCollection
 *
 * @param G4VHitsCollection *hc
 */
void Analysis::ProcessHitCollection(G4VHitsCollection *hc,G4int eventID){

    // Looping through the hit collection
```

[7]C arrays start at 0, so memory is allocated for one more than the total number of layers. This allows for `NUMLAYERS+1` to be used an index into the histogram for the total of all layers in the material (either `gap` or `absorber`).

```
    G4double hitColEDepTot_Abs[NUMLAYERS+1];   // Total EDep (abs) for Hit Collection
    G4double hitColEDepTot_Gap[NUMLAYERS+1];   // Total EDep (gap) for Hit Collection
    G4int PID;                                 // Parent ID
    for(int i= 0; i < NUMLAYERS+1; i++){
        hitColEDepTot_Abs[i] = 0.0;
        hitColEDepTot_Gap[i] = 0.0;
    }

    // Energy Deposition of the event
    for(G4int i = 0; i < hc->GetSize(); i++){
        CaloHit* hit = (CaloHit*) hc->GetHit(i);

        G4double eDep = hit->GetEdep();
        G4int layerNum = hit->GetLayerNumber();
        if (strcmp(hit->GetVolume()->GetName(),"Gap")){
            // Hit occured in the Gap
            hitColEDepTot_Gap[layerNum] += eDep;
            (hHitTotEDepGap[layerNum])->Fill(eDep);
        }else if(strcmp(hit->GetVolume()->GetName(),"Absorber")){
            // Hit occured in the Abs
            hitColEDepTot_Abs[layerNum] += eDep;
            (hHitTotEDepAbs[layerNum])->Fill(eDep);

            /* Is this a secondary electron of the event? */
            if(hit->GetParticle()->GetPDGEncoding() == 11){
                PID = hit->GetParentID();
                if (PID < NUMPID){
                    (hSecElecKinAbs[layerNum][PID])->Fill(hit->GetKineticEnergy());
                }
            }
        }
        else{
            G4cout<<"ERROR - Unkown Volume for sensitive detector"<<G4endl;
        }
    }

    // Adding this Hit collection's energy deposited to event total
    for (int i = 0; i < NUMLAYERS; i++){
        // Incrementing each individual bin
        eventEDepTot_Abs[i] += hitColEDepTot_Abs[i];
        eventEDepTot_Gap[i] += hitColEDepTot_Gap[i];

        // Last bin is Calorimter Total (all Abs layers and all Gap layers)
        eventEDepTot_Abs[NUMLAYERS] += hitColEDepTot_Abs[i];
        eventEDepTot_Gap[NUMLAYERS] += hitColEDepTot_Gap[i];
    }
}
```

Finally, a run macro was written to control the entire run (Listing 31). The material and thickness of the detector are declared (made possible by the use of `DetectorMessenger`), and then the detector is dynamically updated. A ^{60}Co source is simulated by shooting photons of the 1.1732 MeV and 1.3325 MeV. The source particle is then changed to a neutron, and thermal (0.025 eV) neutrons are shot at the detector. The thickness of the absorber is then increased, the geometry updated, and the entire process repeated. As these runs tend to take a large amount of time, GEANT4 was parallelized for use with MPI to take advantage of the cluster computing power.

Listing 31: Run Macro

```
#
/tracking/verbose 0
#
# Setting up the detector
#
/PolymerTransport/det/setAbsMat PS_Detector
/PolymerTransport/det/setGapMat G4_POLYSTYRENE
/PolymerTransport/det/setGapThick 0.3175 cm
#
/PolymerTransport/det/setAbsThick 15 um
/PolymerTransport/det/update
# Cobalt 60
/gun/particle gamma
/gun/direction 0 0 1
/gun/energy 1.1732 MeV
/run/beamOn 500000000 # 500 Million
/gun/energy 1.3325 MeV
/run/beamOn 500000000 # 500 Million
# Neutron
/gun/particle neutron
/gun/energy 0.025 eV
/run/beamOn 1000000 # 1 Million
#
/PolymerTransport/det/setAbsThick 25 um
/PolymerTransport/det/update
```

VI. Results

A. Energy Deposition

The energy deposition was calcutated for neutron and gamma events for films of thickness of 15 μm, 25 μm, 50 μm, 150 μm, 300 μm, 600 μm, 1 mm and 1 cm (Figure 12, 13).

Photons have a very low probability of interacting in the film due to polymer film being a low z-material. This is reflected in the majority of the events not interacting at all; about 1 in 10,000 of the events deposit energy in the film as seen in Figure 12. Several classic features of the spectra are apparent on the 1 cm thick thin. These included the photo-peak in which all of the incident energy of the ^{60}Co is deposited in the film, as well as the individual Compton edges of the two photons fromn ^{60}Co. These features are not visiable on the measured spectra due to the poor energy resolution of these films. There is also physical evidance of a lack of a Compton edge on the thinner films, but the films greater than 150 μmthick show some feature around 0.2 MeV. Films thinner than 150 μmshow a very small amount of energy deposition that quickly tails off for higher energies, indicating that when a photon interaction occurs in the film the electrons from that interaction leave the film and the only energy deposition occurs from small ionizations as the highly energetic electron leaves the film material. It is also observed that the thinnest film (15 μm) has an average energy depostion of around 10 keV, while the 1 cm film has an average energy deposition of around 150 keV. The simulated energy deposition for neutron interactions in thin films is shown in Figure 13. Several features of the spectra can be immediately noted. For thick films (1 cm) there is a very high probability that a given event will deposit all of its energy in the film (as expected). Thinner films have a smaller probability of depositing all of their energy, but this is overshawded by the thick samples when plotted. It is also intresting to note that it is possible to observe the comparative effects of the the α and ^{3}H in the neutron energy depostion spectra. The triton has a much shorter range ($\tilde{1}0$ μmin PS [9]) than the α ($\tilde{6}0$ μm) so it has a higher probability of depositing all of its energy. Thus, for energies above 2.73 MeV (the energy of the triton) there is a higher probability of energy energy deposition (by about a factor of 10). These events are still very infrequent compared to the probability of depositing all of the reaction product energy. Even for the 15 μmthe average energy depostion was above 50% of the total Q-value of the reaction, and by 200 μmthis average energy deposited approaches 95% of the total 4.78 MeV.

B. Secondary Electron Energy Distribuion

The distribution of secondary electrons from photon interactions are plotted in Figure 14. From these results it can be concluded that the it is unlikely (around 1 in 10,000) that an electron will be scattered with the maximum Compton scattering kinetic energy, but rather have an energy somewhat lower than that. The distribution of secondary electrons from photon interactions is actually very flat, implying that it is likely for the electron from a Compton scattering event to have an energy in the 100's of keV. The distribution of the next generation of electrons was also calculated, and this distrubiton was also quite entergetic (with a maximum energy corresponding to 0.55 MeV) but with a much large probability of having a collision that produces and electron with a much lower energy.

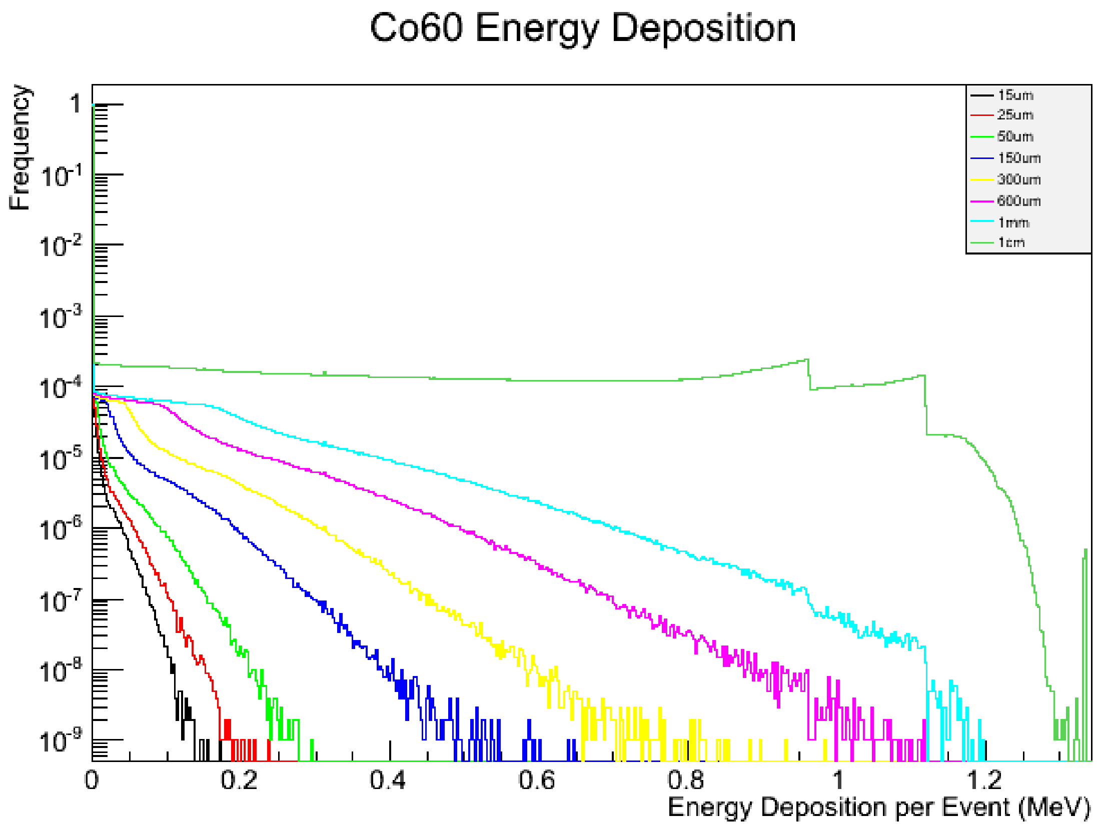


Fig. 12: Simulated Energy Depositon for a Single Film (gammas)

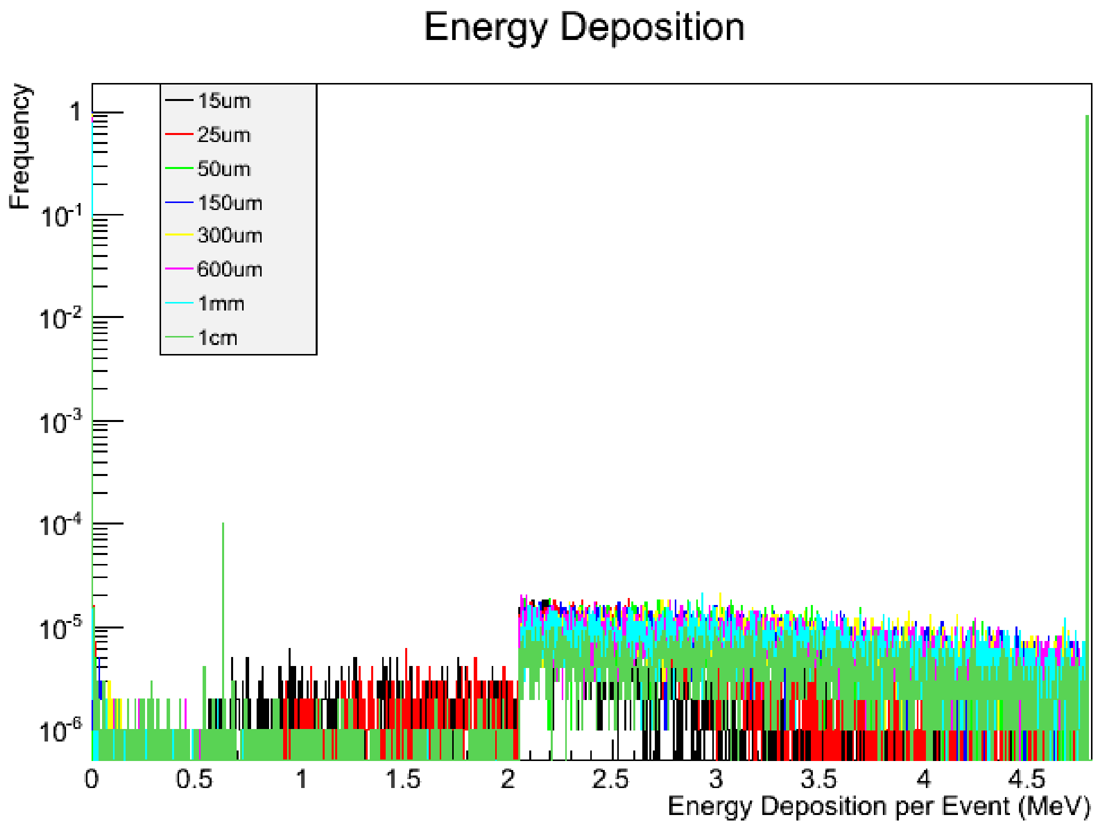
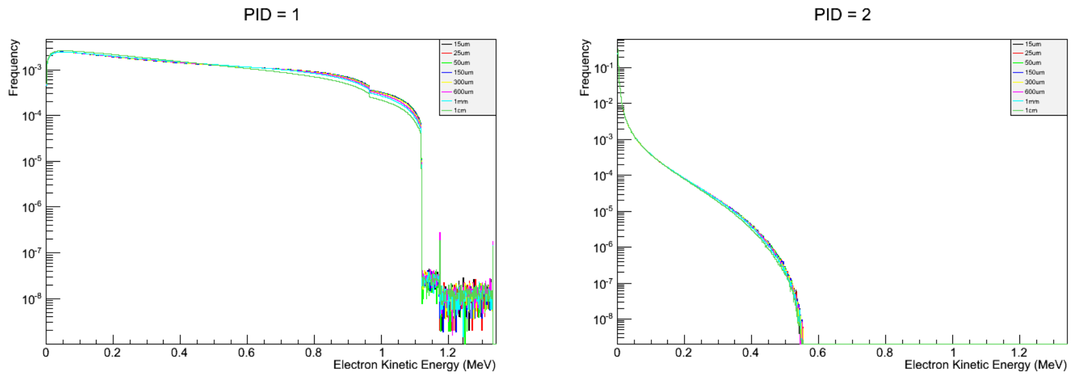


Fig. 13: Simulated Energy Depositon for a Single Film (neutrons)



(a) First Secondary Electron

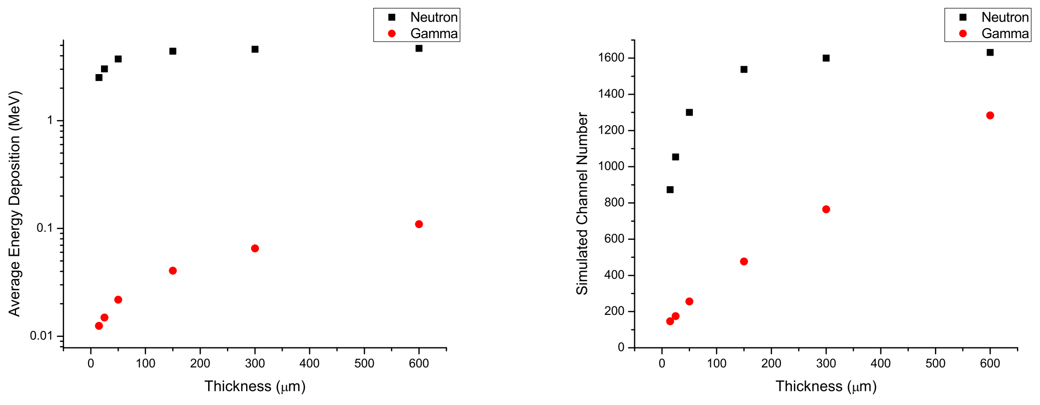
(b) Second Secondary Electron

Fig. 14: Simulated kinetic energies of electrons from ^{60}Co interactions

VII. Conclusions

GEANT4 has been employed to simulate the energy spectra of electrons and energy deposition from thermal neutrons and ^{60}Co gammas. A versitile implemenation of the geometry was used in which it is possible to dynamically set the materials, thickness, and number of layers between runs. In addition, analysis methods have been written to aid in the reporting of the results. This simulation was verified by reproducting the single collision energy loss spectra for water, and also by comparing the average energy deposited to the measured average channel number for film ranging from 15μmto 600μm.

The energy deposition of the films were calculated and plotted in Figure 13 and Figure 12. It is then observable that the gamma interactions have a very low probability of depositing a majority of the energy from a ^{60}Co photon into the material, while neturons tend to deposit over 50% of their energy in the material for a 15μmfilm, and increasing to 96% for a 1 cm thick film. Figure 15 shows the average energy deposition as a function of thickness for neturons and gammas, along with the calculated channel number (according to Equation 4). At thickness of less than 200μmthere is signifcant seperation between the average energy deposited by neutron events compared to gamma events. As the thickness of the films increased the average neturon energy approached the asymptotic limit of 4.78 MeV, while the average gamma energy increased. This creates less seperation between the two, and provides less of an ability for neutron-gamma discrimination based on pulse height.



(a) Energy Deposition

(b) Esimated Channel Number

Fig. 15: Comparison between average neturon and gamma energy deposition

REFERENCES

[1] J. E. Turner, H. G. Paretzke, R. N. Hamm, H. A. Wright, and R. H. Ritchie, "Comparative study of electron energy deposition and yields in water in the liquid and vapor phases," *Radiation Research*, vol. 92, pp. 47–60, Oct. 1982. ArticleType: research-article / Full publication date: Oct., 1982 / Copyright 1982 Radiation Research Society.

[2] S. Agostinelli, J. Allison, K. Amako, and et al., "Geant4a simulation toolkit," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 506, pp. 250–303, July 2003.

[3] G. Collaboration, "Geant4 user's guide for application developers." http://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/ForApplicationDeveloper/html/in Dec. 2011. Version geant4 9.5.0.

[4] J. Allison, K. Amako, J. Apostolakis, and et al., "Geant4 developments and applications," *Nuclear Science, IEEE Transactions on*, vol. 53, pp. 270–278, Feb. 2006.

[5] CERN, "Physics lists EM constructors in geant4 9.3." http://geant4.cern.ch/geant4/collaboration/working_groups/electromagnetic/physlist9.3.shtml, Feb. 2012.

[6] CERN, "Reference physics lists." http://geant4.cern.ch/support/proc_mod_catalog/physics_lists/referencePL.shtml, Oct. 2008.

[7] CERN, "Detector defination and response." http://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/ForApplicationDeveloper/html/ch04s04.html, 2012.

[8] J. B. Birks, "Scintillations from organic crystals: Specific fluorescence and relative response to different radiations," *Proceedings of the Physical Society. Section A*, vol. 64, pp. 874–877, Oct. 1951.

[9] H. Kudo and K. Tanaka, "Recoil ranges of 2.73 MeV tritons and yields of 18F produced by the 16O(t,n)18F reaction in neutron-irradiated lithium compounds containing oxygen," *The Journal of Chemical Physics*, vol. 72, no. 5, p. 3049, 1980.