

Energy Deposition in Polymers

Matthew J. Urffer

TODO LIST

Expand this section 10

CONTENTS

5			
6	I	Previous Work	6
7	I-A	Spectra Measurements	6
8	I-B	Single Collision Energy Loss	7
9	II	Introduction to GEANT4	10
10	II-A	Organization of the GEANT4 Toolkit	10
11	II-B	GEANT4 Tracking and Secondaries	10
12	V	Methods	28
13	V-A	GEANT4 Implementation	28
14		V-A1 Detector Geometry	28
15		V-A2 Physics Lists	30
16		V-A3 Primary Event Generator	32
17	V-B	Sensitive Detectors and Hits	33
18	V-C	Analysis	36
19	V-D	Determination of Energy Deposition	37
20	IV	Simulation Validation	24
21	IV-A	Energy Deposition Validation	24
22	IV-B	Spectra Validation	25
23	V	Methods	28
24	V-A	GEANT4 Implementation	28
25		V-A1 Detector Geometry	28
26		V-A2 Physics Lists	30
27		V-A3 Primary Event Generator	32
28	V-B	Sensitive Detectors and Hits	33
29	V-C	Analysis	36
30	V-D	Determination of Energy Deposition	37
31	VI	Results	40
32	VI-A	Energy Deposition	40
33	VI-B	Secondary Electron Energy Distribuion	40
34	VII	Conclusions	43
35		References	44

LIST OF FIGURES

36

37	1	Spectra properties as a function of film thickness	6
38	2	Gamma intrinsic efficiency (dashed lines) plotted against neutron counts (solid)	7
39	3	Single-collision energy loss spectra for electrons in water [1]	8
40	4	Average and median energy transfer in liquid water as functions of incident-electron energy [1]	9
41	10	World, Calorimeter, Layer and Absorber and Gap	30
42	11	10 Layer Detector with a simulated gamma event	31
43	7	Single Collision Energy Loss of Water	24
44	8	Gamma Simulation Agreement	26
45	9	Neutron Simulation Agreement	27
46	10	World, Calorimeter, Layer and Absorber and Gap	30
47	11	10 Layer Detector with a simulated gamma event	31
48	12	Simulated Energy Depositon for a Single Film (gammas)	41
49	13	Simulated Energy Depositon for a Single Film (neutrons)	42
50	14	Simulated kinetic energies of electrons from ^{60}Co interactions	42
51	15	Comparison between average neturon and gamma energy deposition	43

LIST OF TABLES

LISTINGS

53

54	1	Tracking Example	11
55	17	World Physical Volume	28
56	18	Calorimeter Volume	29
57	19	Layer Volume	29
58	20	Absorber and Gap Volumes	29
59	21	Implemented Physics List	31
60	22	Implemented Physics List	32
61	23	Primary Event Generator	32
62	24	Generate Primaries	33
63	25	Calorimeter Hit	33
64	26	Sensitive Detector	35
65	27	Creating Sensitive Detectors	36
66	28	Event Action	36
67	29	Run Action	37
68	30	Process Hit Collection	37
69	31	Run Macro	39
70	17	World Physical Volume	28
71	18	Calorimeter Volume	29
72	19	Layer Volume	29
73	20	Absorber and Gap Volumes	29
74	21	Implemented Physics List	31
75	22	Implemented Physics List	32
76	23	Primary Event Generator	32
77	24	Generate Primaries	33
78	25	Calorimeter Hit	33
79	26	Sensitive Detector	35
80	27	Creating Sensitive Detectors	36
81	28	Event Action	36
82	29	Run Action	37
83	30	Process Hit Collection	37
84	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% ^6LiF , 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

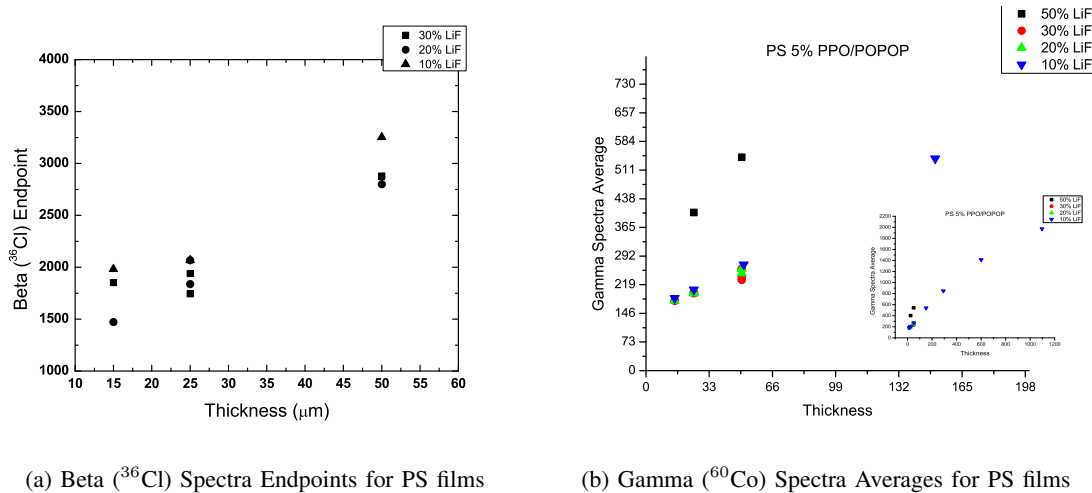


Fig. 1: Spectra properties as a function of film thickness

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 rates¹ while minimizing the response of the detector to photons.

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

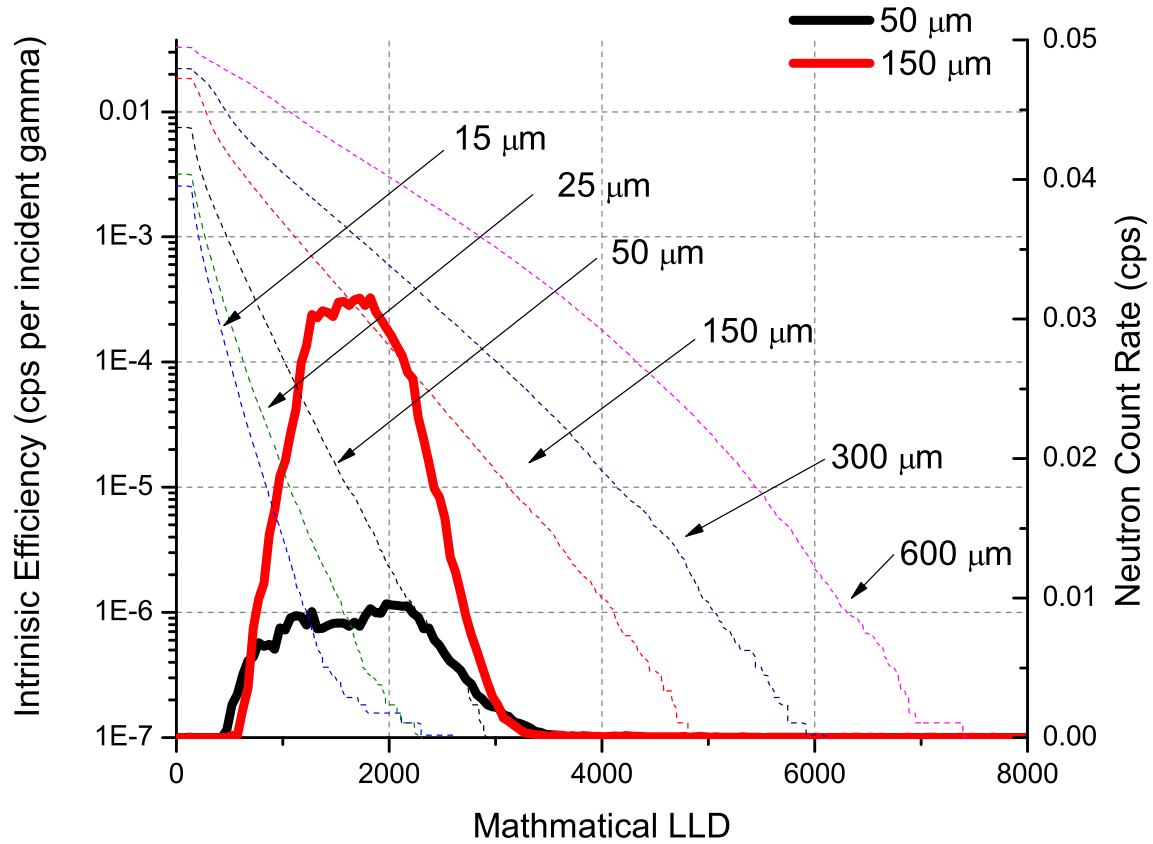


Fig. 2: Gamma intrinsic efficiency (dashed lines) plotted against neutron counts (solid)

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 ${}^6\text{Li}$ 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

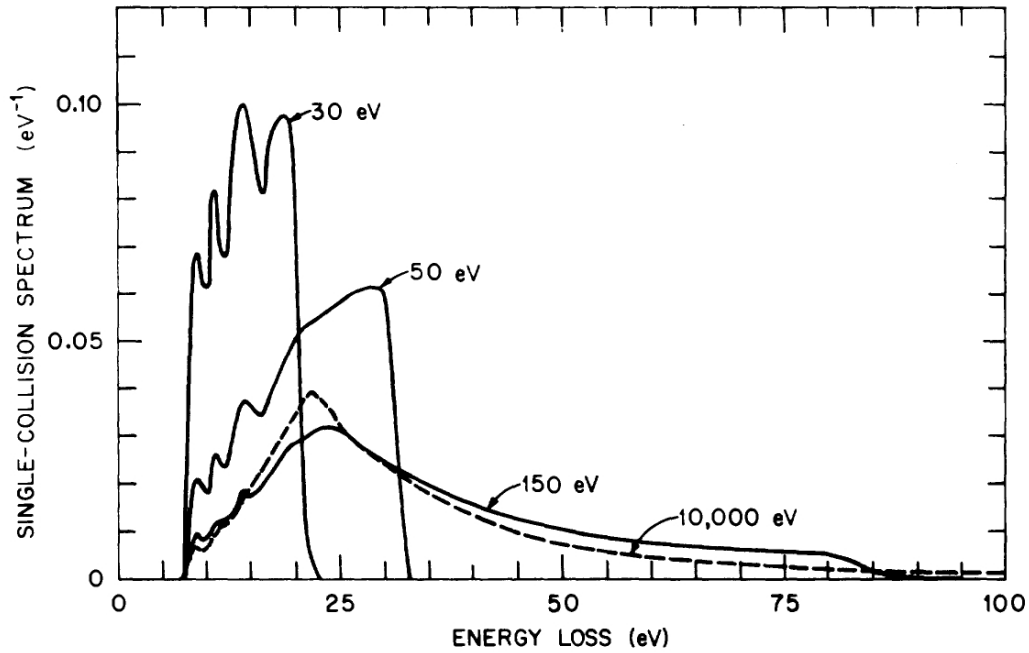


Fig. 3: Single-collision energy loss spectra for electrons in water [1]

114 while a few collisions result in large energy transfers most of the collisions do not. It is also interesting to note that
 115 the average and median do not increase linearly with the incident energy past 100 eV (the ordinate axis is a log
 116 scale). In fact, the average energy transferred per collision is mostly bounded by 60 eV even for incident electron
 117 energies of 10 keV. This is significant because it implies that high energy electrons from photon events will deposit
 118 a small fraction of their energy in the material.

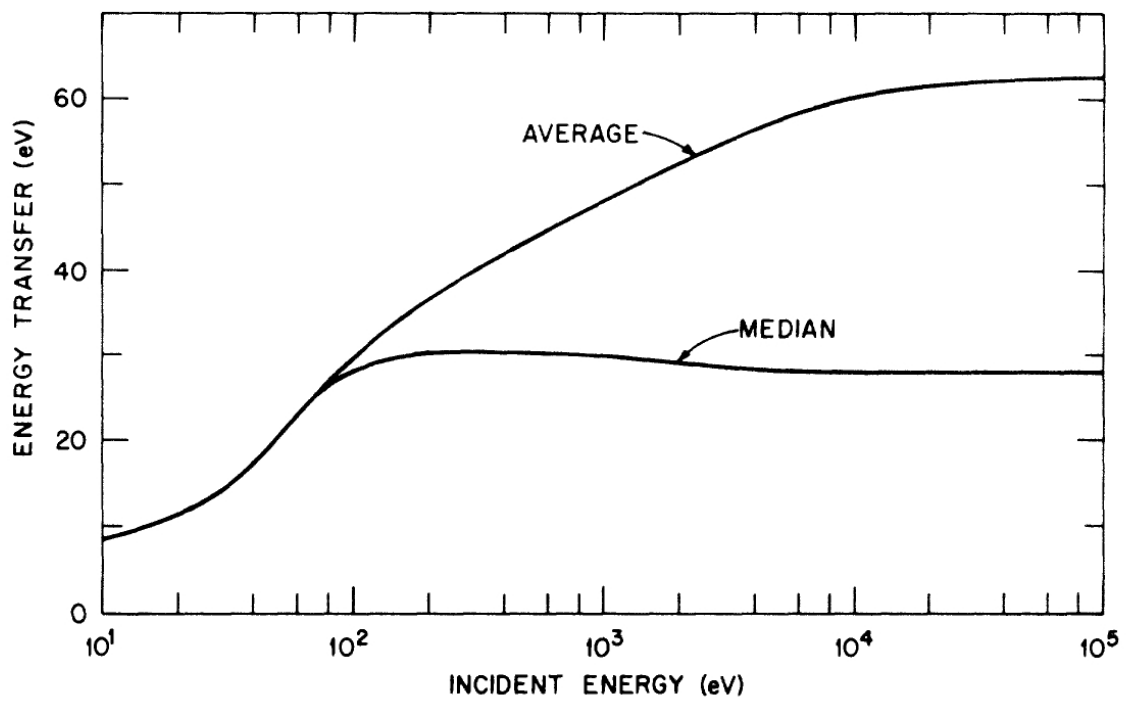


Fig. 4: Average and median energy transfer in liquid water as functions of incident-electron energy [1]

II. INTRODUCTION TO GEANT4

GEANT4 (GEometry 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 existing 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). 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
- `G4VUserPrimaryGeneratorAction` which defines the generation of primary events.

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

- `G4UserRunAction` which allows for user actions

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
168      0         0         0   -6.59   2.5e-08         0         0      Absorber  initStep
169      1         0         0   -3.64         0         2.95      2.95      Absorber  NeutronInelastic
170      :----- List of 2ndaries --- #SpawnInStep= 2(Rest= 0,Along= 0,Post= 2), #SpawnTotal= 2 -----
171      :         0         0   -3.64     2.73          triton
172      :         0         0   -3.64     2.05          alpha
173      :----- EndOf2ndaries  Info -----
174
*****
* 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
179      0         0         0   -3.64     2.05         0         0      Absorber  initStep
180      1 -0.000201  0.000128   -3.64     2.01   0.0491  0.000266  0.000266      Absorber  ionIoni
181      2 -0.00049  0.000312   -3.64     1.93   0.0705  0.000381  0.000647      Absorber  ionIoni
182
183
*****
* 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
188      0         0         0   -3.64     2.73         0         0      Absorber  initStep
189      1 0.000339 -0.000215   -3.64     2.71   0.0116  0.000447  0.000447      Absorber  hIoni
190

```

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 wiki². 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
```

²It 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`.

```

223 2    worldS = new G4Box("World",worldSizeXY, worldSizeXY, worldSizeZ*0.5);
224    worldLV = new G4LogicalVolume(worldS,defaultMaterial,"World");
225 4    worldPV = new G4PVPlacement(0,G4ThreeVector(),worldLV,"World",0,false,0,fCheckOverlaps);
226

```

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

Listing 3: Calorimeter Volume

```

230
231 // Calorimeter (gap material)
232 2    caloS = new G4Tubs("Calorimeter",iRadius,oRadius,caloThickness/2,startAngle,spanAngle);
233    caloLV = new G4LogicalVolume(caloS,gapMaterial,"Calorimeter");
234 4    caloPV = new G4PVPlacement(0,G4ThreeVector(),caloLV,"Calorimeter",worldLV,false,0,
235    fCheckOverlaps);
236

```

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

Listing 4: Layer Volume

```

241
242 1 // Layer (Consists of Absorber and Gap)
243    layerS = new G4Tubs("Layer",iRadius,oRadius,layerThickness/2,startAngle,spanAngle);
244 3    layerLV = new G4LogicalVolume(layerS,defaultMaterial,"Layer");
245    if (nofLayers > 1){
246 5        layerPV = new G4PVReplica("Layer",layerLV,caloLV,kZAxis,nofLayers,layerThickness,-
247        caloThickness/2);
248    }else{
249 7        layerPV = new G4PVPlacement(0,G4ThreeVector(0.0,0.0,0.0),layerLV,"Layer",caloLV,false,0,
250        fCheckOverlaps);
251    }
252

```

253 Finally, the neutron absorber and gap material were defined as single cylinders which were then placed in the
 254 layer mother volume (Listing 20). The size of these solids (and the materials) could be set either at compile time
 255 through DetectorConstruction constructor or by using the DetectorMessenger in the run macro. Figure
 256 11 shows a rendering of the 10 layers of the detector with the trajectories from a gamma event.

Listing 5: Absorber and Gap Volumes

```

257
258 // Absorber
259 2    absS = new G4Tubs("Abso",iRadius,oRadius,absThickness/2,startAngle,spanAngle);
260    absLV = new G4LogicalVolume(absS,absMaterial,"Absorber",0);
261 4    absPV = new G4PVPlacement(0,G4ThreeVector(0.0,0.0,-gapThickness/2),absLV,"Absorber",layerLV,
262    false,0,fCheckOverlaps);
263
264 6 // Gap

```

```

265 gapS = new G4Tubs("Gap", iRadius, oRadius, gapThickness/2, startAngle, spanAngle);
266 gapLV = new G4LogicalVolume(gapS, gapMaterial, "Gap", 0);
267 gapPV = new G4PVPlacement(0, G4ThreeVector(0.0, 0.0, absThickness/2), gapLV, "Gap", layerLV, false
268 , 0, fCheckOverlaps);
269

```

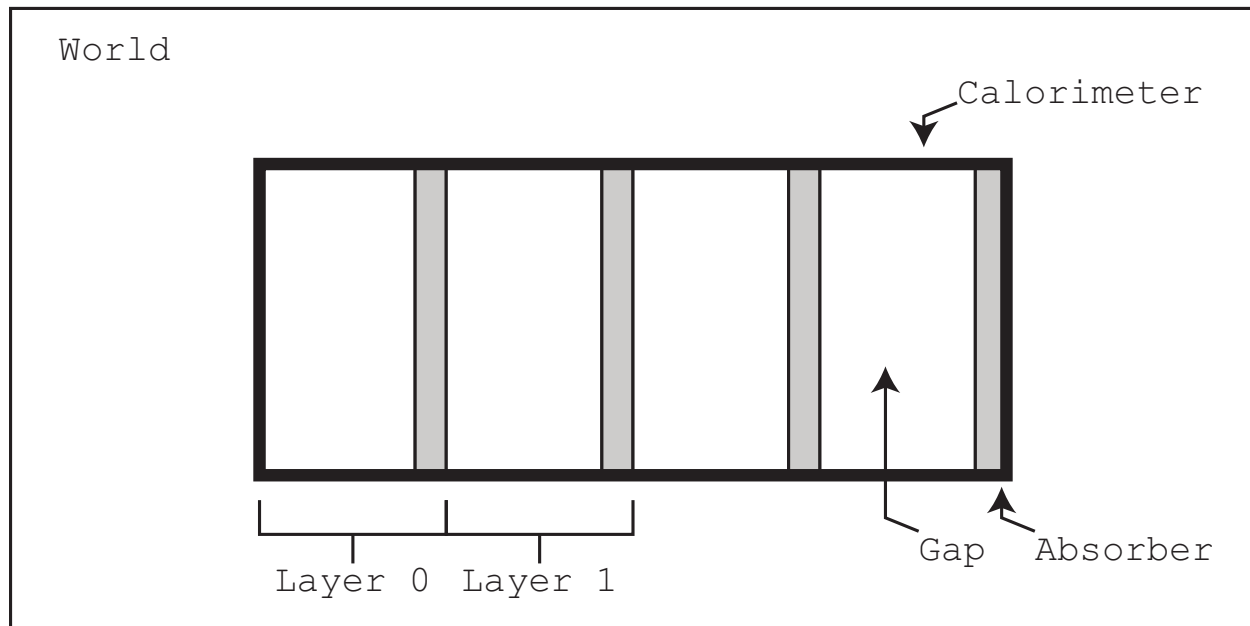


Fig. 5: World, Calorimeter, Layer and Absorber and Gap

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 combining 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 `EMStandardPhysics` 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].

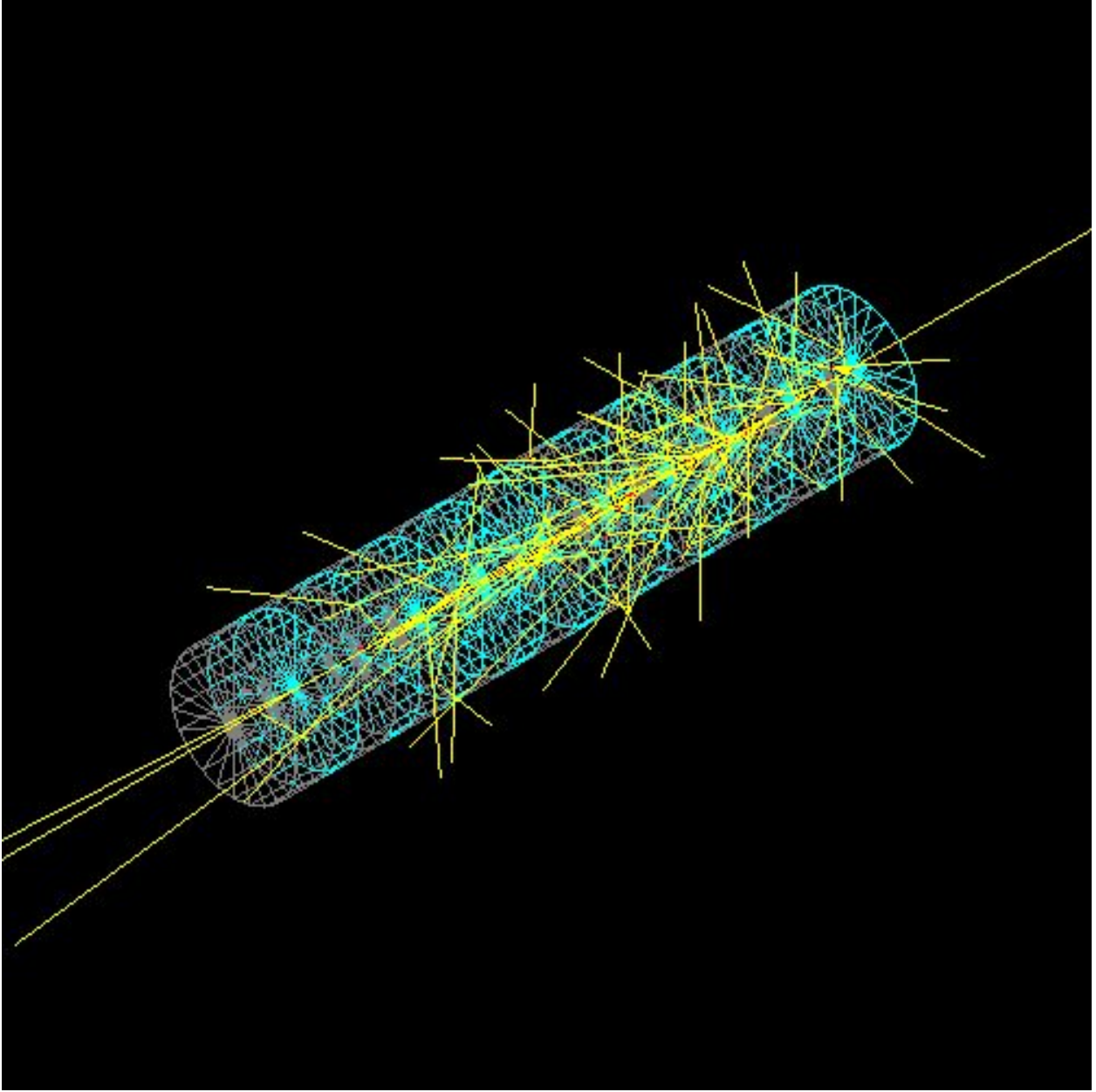


Fig. 6: 10 Layer Detector with a simulated gamma event

- `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 ${}^6\text{Li}(n, \alpha){}^3\text{H}$ interaction the `G4IonPhysics` list was used.

Listing 6: Implemented Physics List

```

292
293 1  /**
294     * PhysicsList
295 3  *
296     * Constructs the physics of the simulation
297 5  */
298 PhysicsList::PhysicsList() : G4VModularPhysicsList() {
299 7     currentDefaultCut    = 10*nm;
300
301 9     // Adding Physics List
302     //RegisterPhysics( new G4EmDNAPhysics());
303 11    RegisterPhysics( new G4EmStandardPhysics());
304     RegisterPhysics( new G4EmLivermorePhysics());
305 13    RegisterPhysics( new HadronPhysicsQGSP_BERT_HP());
306     RegisterPhysics( new G4IonPhysics());
307 15 }
308

```

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

Listing 7: Implemented Physics List

```

310
311 1 void PhysicsList::SetCuts() {
312     SetDefaultCutValue(10*nm);
313 3 }
314

```

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

Listing 8: Primary Event Generator

```

319
320 PrimaryGeneratorAction::PrimaryGeneratorAction() : G4VUserPrimaryGeneratorAction(), fParticleGun
321     (0) {
322 2     G4int nofParticles = 1;
323     fParticleGun = new G4ParticleGun(nofParticles);
324 4
325     // default particle kinematic
326 6     G4ParticleDefinition* particleDefinition = G4ParticleTable::GetParticleTable()->FindParticle("e
327         -");
328     fParticleGun->SetParticlePosition(G4ThreeVector(0.,0.,0.0));
329 8     fParticleGun->SetParticleDefinition(particleDefinition);
330     fParticleGun->SetParticleMomentumDirection(G4ThreeVector(0.,0.,1.));
331 10    fParticleGun->SetParticleEnergy(50.*MeV);
332 }
333

```

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

Listing 9: Generate Primaries

```

336
337 void PrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
338 {
339     // This function is called at the begining of event
340
341     // In order to avoid dependence of PrimaryGeneratorAction
342     // on DetectorConstruction class we get world volume
343     // from G4LogicalVolumeStore
344     G4double worldZHalfLength = 0;
345     G4LogicalVolume* worlLV = G4LogicalVolumeStore::GetInstance()->GetVolume("World");
346     G4Box* worldBox = 0;
347     if ( worlLV) worldBox = dynamic_cast< G4Box*>(worlLV->GetSolid());
348     if ( worldBox ) {
349         worldZHalfLength = worldBox->GetZHalfLength();
350     }
351     else {
352         G4cerr << "World volume of box not found." << G4endl;
353         G4cerr << "Perhaps you have changed geometry." << G4endl;
354         G4cerr << "The gun will be place in the center." << G4endl;
355     }
356
357     // Set gun position
358     fParticleGun->SetParticlePosition(G4ThreeVector(0., 0., -worldZHalfLength+1*cm));
359     fParticleGun->GeneratePrimaryVertex(anEvent);
360 }
361

```

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

```

376
377  /**
378 2  * @brief - Hit: a snapshot of the physical interaction of a track in the sensitive region of a
379      detector
380      *
381 4  * Contains:
382      * - Particle Information (type and rank (primary, secondary, tertiary ...))
383 6  * - Position and time
384      * - momentum and kinetic energy
385 8  * - deposition in volume
386      * - geometric information
38710 */
388 class CaloHit : public G4VHit {
38912 public:
390     CaloHit(const G4int layer);
39114     ~CaloHit();
392
39316     inline void* operator new(size_t);
394     inline void operator delete(void*);
39518     void Print();
396
39720 private:
398     G4double edep;           /* Energy Deposited at the Hit */
39922     G4ThreeVector pos;       /* Position of the hit */
400     G4double stepLength;     /* Step Length */
40124     G4ThreeVector momentum; /* Momentum of the step */
402     G4double kEnergy;        /* Kinetic Energy of the particle */
40326     G4int trackID;          /* Track ID */
404     G4int parentID;          /* Parent ID */
40528     G4ParticleDefinition* particle; /* Particle Definition */
406     G4int particleRank;      /* Primary, Secondary, etc */
40730     G4VPhysicalVolume* volume; /* Physical Volume */
408     G4int layerNumber;       /* Copy Number of Layer */
40932
410     public:
41134         // Setter and Getters
412 };
41336
414 typedef G4THitsCollection<CaloHit> CaloHitsCollection;
41538 extern G4Allocator<CaloHit> HitAllocator;
416
41740 inline void* CaloHit::operator new(size_t){
418     void *aHit;
41942     aHit = (void *) HitAllocator.MallocSingle();
420     return aHit;
42144 }
422
42346 inline void CaloHit::operator delete(void *aHit){

```

```

424 HitAllocator.FreeSingle((CaloHit*) aHit);
425 }
426

```

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

Listing 11: Sensitive Detector

```

429 /**
430  * ProcessHits
431  *
432  *
433  * Adds a hit to the sensitive detector, depending on the step
434  */
435 G4bool CaloSensitiveDetector::ProcessHits(G4Step* aStep,G4TouchableHistory*) {
436
437     G4double edep = aStep->GetTotalEnergyDeposit();
438     G4double stepLength = aStep->GetStepLength();
439
440     // Getting the copy number
441     G4TouchableHistory* touchable = (G4TouchableHistory*)
442         (aStep->GetPreStepPoint()->GetTouchable());
443     G4int layerIndex = touchable->GetReplicaNumber(1);
444
445     // Creating the hit
446     CaloHit* newHit = new CaloHit(layerIndex);
447     newHit->SetTrackID(aStep->GetTrack()->GetTrackID());
448     newHit->SetParentID(aStep->GetTrack()->GetParentID());
449     newHit->SetEdep(edep);
450     newHit->SetStepLength(stepLength);
451     newHit->SetPosition(aStep->GetPreStepPoint()->GetPosition());
452     newHit->SetLayerNumber(layerIndex);
453     newHit->SetMomentum(aStep->GetPreStepPoint()->GetMomentum());
454     newHit->SetKineticEnergy(aStep->GetPreStepPoint()->GetKineticEnergy());
455     newHit->SetParticle(aStep->GetTrack()->GetDefinition());
456     newHit->SetVolume(aStep->GetTrack()->GetVolume());
457
458     // Adding the hit to the collection
459     hitCollection->insert( newHit );
460
461     return true;
462 }
463

```

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

Listing 12: Creating Sensitive Detectors

```

469
470 1 /**
471     * SetSensitiveDetectors
472 3  *
473     * Setting the Sensitive Detectors of the Detector
474 5  */
475 void DetectorConstruction::SetSensitiveDetectors() {
476 7     G4SDManager* SDman = G4SDManager::GetSDMpointer();
477     absSD = new CaloSensitiveDetector("SD/AbsSD", "AbsHitCollection");
478 9     SDman->AddNewDetector(absSD);
479     absLV->SetSensitiveDetector(absSD);
480 11
481     gapSD = new CaloSensitiveDetector("SD/GapSD", "GapHitCollection");
482 13     SDman->AddNewDetector(gapSD);
483     gapLV->SetSensitiveDetector(gapSD);
484 15 }
485

```

486 C. Analysis

487 Analysis of hit collection was preformed with ROOT. Once again there are other options (notably OpenScientist)

488 but previous experience was why ROOT was selected as the base for the Analysis framework. A singleton class

489 was written for the analysis which processed the hit collections, assigning the various results to root histograms.

490 User action classes EventAction and RunAction are called at the beginning and end of each run and event,

491 respectively (Listing 28,29). These classes allowed for the analysis code to be independent of the simulation.

Listing 13: Event Action

```

492
493 1 EventAction::EventAction() : G4UserEventAction() {
494     // Nothing to be Done Here
495 3 }
496
497 5 /**
498     * BeginOfEventAction
499 7  *
500     * @param const G4Event* event - event to be processed
501 9  *
502     * At the begining of an event we want to clear all the event
503 11 * accumulation variables.
504     */
505 13 void EventAction::BeginOfEventAction(const G4Event* event) {
506     Analysis::GetInstance()->PrepareNewEvent(event);
507 15 }
508
509 17 /**
510     * EndOfEventAction
511 19  *
512     * @param const G4Event* event - event to be processed

```

```

51321 *
514 * At the end of an event we want to call analysis to process
51523 * this event, and record the useful information.
516 */
51725 void EventAction::EndOfEventAction(const G4Event* event) {
518     Analysis::GetInstance()->EndOfEvent(event);
51927 }
520

```

Listing 14: Run Action

```

521
5221 RunAction::RunAction() : G4UserRunAction() { }
523
5243 void RunAction::BeginOfRunAction(const G4Run* run) {
525     G4cout<<"Starting run: " << run->GetRunID()<< G4endl;
5265     Analysis::GetInstance()->PrepareNewRun(run);
527 }
5287
529 void RunAction::EndOfRunAction(const G4Run* aRun) {
5309     Analysis::GetInstance()->EndOfRun(aRun);
531 }
532

```

533 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.

$$E_{\text{dep,event}} = \sum E_{\text{dep,hit}} \quad (1)$$

534 ProcessHitCollection is called at the end of each event (Listing 30). Each hit is accessed and the layer at
535 which it occurs is determined³. In addition the name of the volume is determined, and the energy deposition of the
536 hit is added to the energy deposition of the event. If the hit occurred in the `absorber` layer and the particle is an
537 electron the kinetic energy of that hit is also recorded.

Listing 15: Process Hit Collection

```

538
5391 /**
540 * ProcessHitCollection
5413 *
542 * @param G4VHitsCollection *hc
5435 */
544 void Analysis::ProcessHitCollection(G4VHitsCollection *hc,G4int eventID){
5457
546     // Looping through the hit collection

```

³C 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 as an index into the histogram for the total of all layers in the material (either gap or absorber).

```

547 9      G4double hitColEdepTot_Abs[NUMLAYERS+1];    // Total EDep (abs) for Hit Collection
548      G4double hitColEdepTot_Gap[NUMLAYERS+1];    // Total EDep (gap) for Hit Collection
54911     G4int PID;                                   // Parent ID
550     for(int i= 0; i < NUMLAYERS+1; i++){
55113         hitColEdepTot_Abs[i] = 0.0;
552         hitColEdepTot_Gap[i] = 0.0;
55315     }
554
55517     // Energy Deposition of the event
556     for(G4int i = 0; i < hc->GetSize(); i++){
55719         CaloHit* hit = (CaloHit*) hc->GetHit(i);
558
55921         G4double eDep = hit->GetEdep();
560         G4int layerNum = hit->GetLayerNumber();
56123         if (strcmp(hit->GetVolume()->GetName(),"Gap")){
562             // Hit occurred in the Gap
56325             hitColEdepTot_Gap[layerNum] += eDep;
564             (hHitTotEdepGap[layerNum])->Fill(eDep);
56527         }else if (strcmp(hit->GetVolume()->GetName(),"Absorber")){
566             // Hit occurred in the Abs
56729             hitColEdepTot_Abs[layerNum] += eDep;
568             (hHitTotEdepAbs[layerNum])->Fill(eDep);
56931
570             /* Is this a secondary electron of the event? */
57133             if(hit->GetParticle()->GetPDGEncoding() == 11){
572                 PID = hit->GetParentID();
57335                 if (PID < NUMPID){
574                     (hSecElecKinAbs[layerNum][PID])->Fill(hit->GetKineticEnergy());
57537                 }
576             }
57739         }
578         else{
57941             G4cout<<"ERROR - Unkown Volume for sensitive detector"<<G4endl;
580         }
58143     }
582
58345     // Adding this Hit collection's energy deposited to event total
584     for (int i = 0; i < NUMLAYERS; i++){
58547         // Incrementing each individual bin
586         eventEdepTot_Abs[i] += hitColEdepTot_Abs[i];
58749         eventEdepTot_Gap[i] += hitColEdepTot_Gap[i];
588
58951         // Last bin is Calorimter Total (all Abs layers and all Gap layers)
590         eventEdepTot_Abs[NUMLAYERS] += hitColEdepTot_Abs[i];
59153         eventEdepTot_Gap[NUMLAYERS] += hitColEdepTot_Gap[i];
592     }
59355 }
594

```

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

```

601 #
602 1 #
603 /tracking/verbose 0
604 3 #
605 # Setting up the detector
606 5 #
607 /PolymerTransport/det/setAbsMat PS_Detector
608 7 /PolymerTransport/det/setGapMat G4_POLYSTYRENE
609 /PolymerTransport/det/setGapThick 0.3175 cm
610 9 #
611 /PolymerTransport/det/setAbsThick 15 um
612 11 /PolymerTransport/det/update
613 # Cobalt 60
614 13 /gun/particle gamma
615 /gun/direction 0 0 1
616 15 /gun/energy 1.1732 MeV
617 /run/beamOn 500000000 # 500 Million
618 17 /gun/energy 1.3325 MeV
619 /run/beamOn 500000000 # 500 Million
620 19 # Neutron
621 /gun/particle neutron
622 21 /gun/energy 0.025 eV
623 /run/beamOn 1000000 # 1 Million
624 23 #
625 /PolymerTransport/det/setAbsThick 25 um
626 25 /PolymerTransport/det/update
627

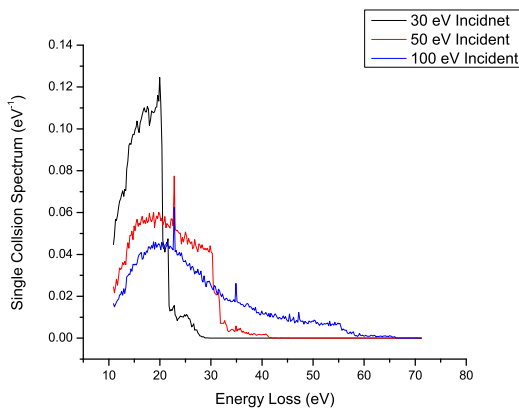
```

IV. SIMULATION VALIDATION

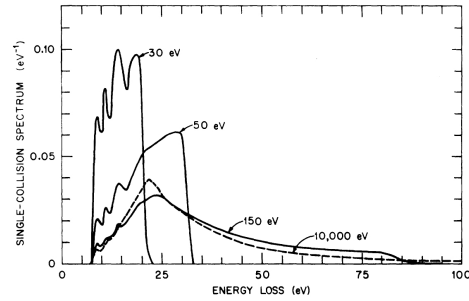
GEANT4 is a toolkit implemented by the user so extensive efforts were completed in order to validate the results and ensure no bugs exists. First steps were taken (for small runs) to compute the energy deposition for small runs by hand in order to make sure they agreed with the analysis code. In addition the reaction products of the ${}^6\text{Li}(n, \alpha){}^3\text{H}$ were checked to make sure that they agreed with the published values⁴. 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 water⁵. The PhysicsList was extended to include G4DNAPhysics and the detector material was set to the NIST definition contained in the toolkit with `G4Material* H2O = 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

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

⁵An 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-_G4DNAIonisation" | awk '{print $5}'`

642 B. Spectra Validation

The simulated energy deposition is not the directly equivalent 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 constants, the energy deposition can be considered equivalent to the scintillation and representative of the measured spectra. Rather than attempting to back out these scaling constants 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 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 evidence that the cascade electrons from the Compton electron are energetic 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).

$$\langle E \rangle = \frac{\int_0^\infty \phi(E) E dE}{\int_0^\infty \phi(E) dE} \text{ where} \quad (2)$$

$$\langle \mu \rangle = \frac{\int_0^\infty f(x) x dx}{\int_0^\infty f(x) dx} \text{ where} \quad (3)$$

643

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 separately for neutrons and gammas.

$$\eta = \sum_t \frac{\langle E \rangle}{\langle \mu \rangle} \quad (4)$$

Fig. 8: Gamma Simulation Agreement

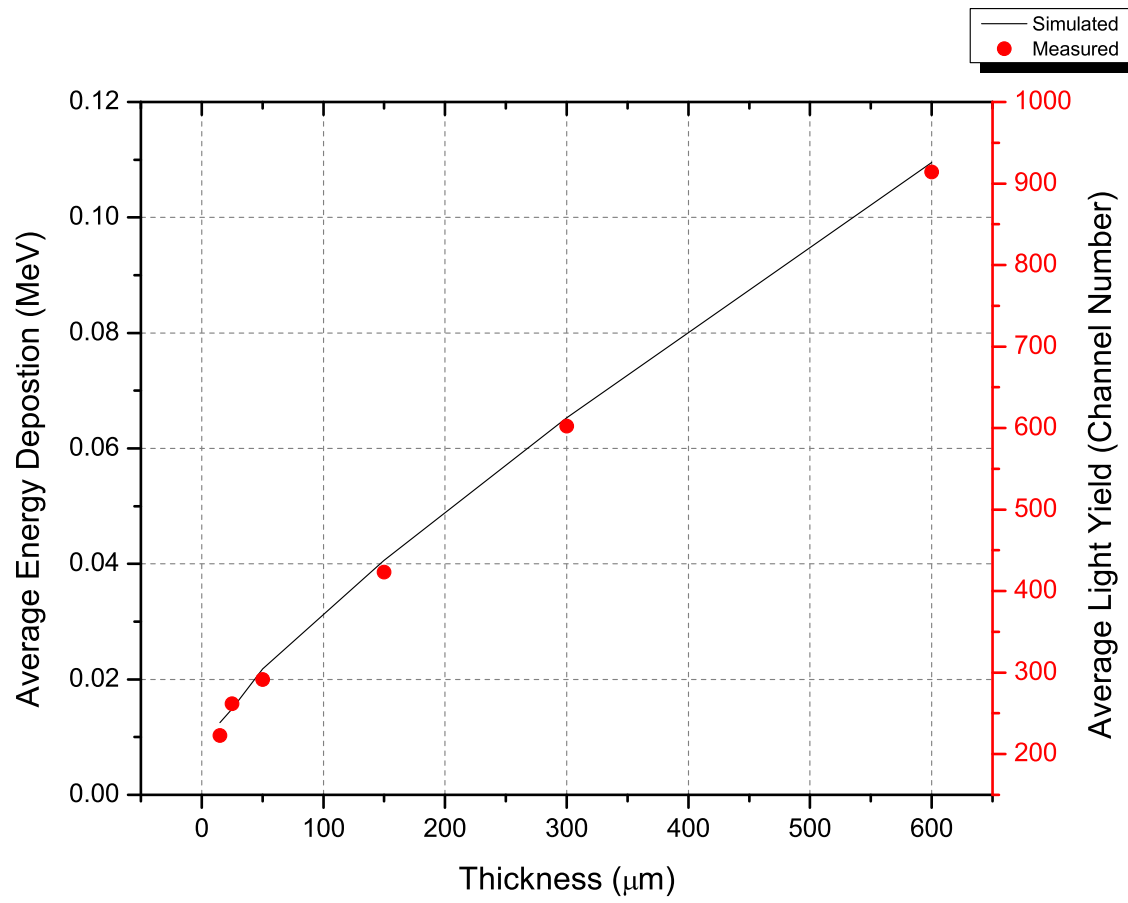
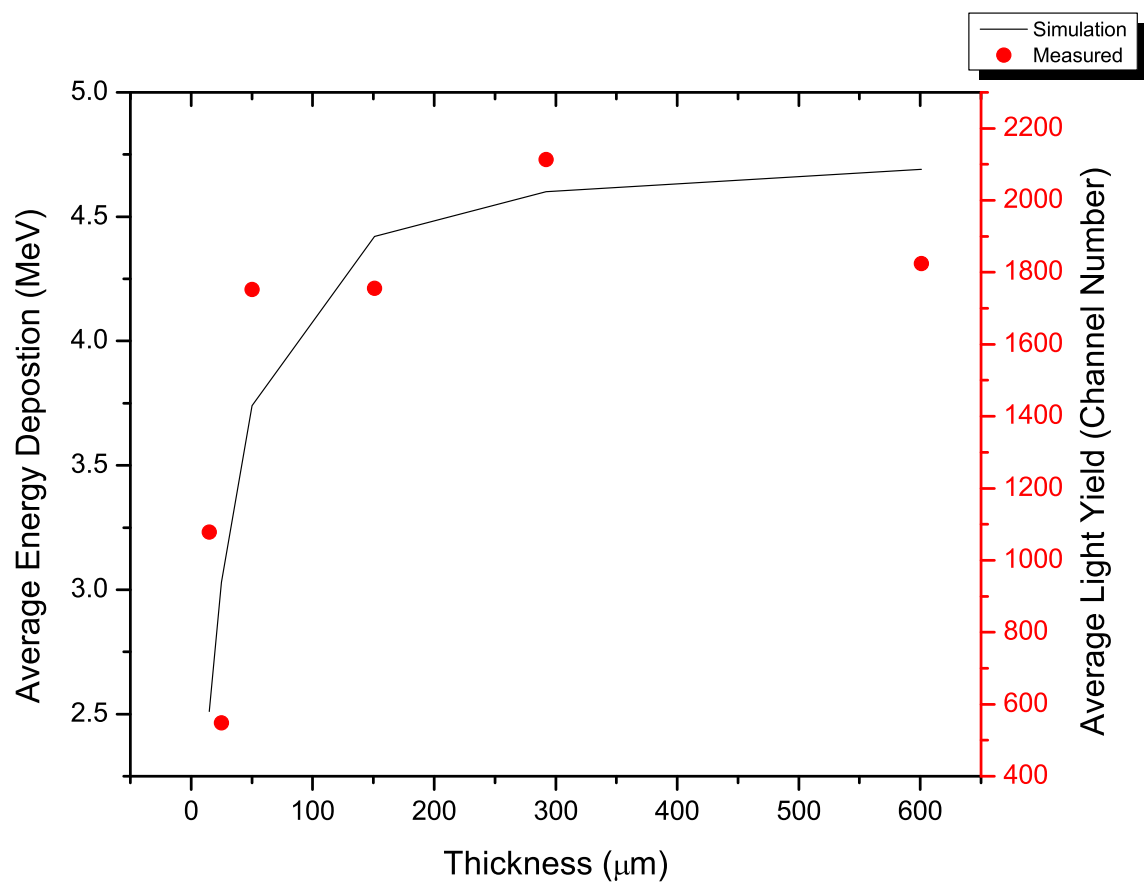


Fig. 9: Neutron Simulation Agreement



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 wiki⁶. 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
```

⁶It 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`.

```

675 worldS = new G4Box("World",worldSizeXY, worldSizeXY, worldSizeZ*0.5);
676 worldLV = new G4LogicalVolume(worldS,defaultMaterial,"World");
677 worldPV = new G4PVPlacement(0,G4ThreeVector(),worldLV,"World",0,false,0,fCheckOverlaps);
678

```

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

Listing 18: Calorimeter Volume

```

682 // Calorimeter (gap material)
683
684 caloS = new G4Tubs("Calorimeter",iRadius,oRadius,caloThickness/2,startAngle,spanAngle);
685 caloLV = new G4LogicalVolume(caloS,gapMaterial,"Calorimeter");
686 caloPV = new G4PVPlacement(0,G4ThreeVector(),caloLV,"Calorimeter",worldLV,false,0,
687 fCheckOverlaps);
688

```

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

Listing 19: Layer Volume

```

693 // Layer (Consists of Absorber and Gap)
694
695 layerS = new G4Tubs("Layer",iRadius,oRadius,layerThickness/2,startAngle,spanAngle);
696 layerLV = new G4LogicalVolume(layerS,defaultMaterial,"Layer");
697 if (nofLayers > 1){
698 layerPV = new G4PVReplica("Layer",layerLV,caloLV,kZAxis,nofLayers,layerThickness,-
699 caloThickness/2);
700 }else{
701 layerPV = new G4PVPlacement(0,G4ThreeVector(0.0,0.0,0.0),layerLV,"Layer",caloLV,false,0,
702 fCheckOverlaps);
703 }
704

```

705 Finally, the neutron absorber and gap material were defined as single cylinders which were then placed in the
706 layer mother volume (Listing 20). The size of these solids (and the materials) could be set either at compile time
707 through DetectorConstruction constructor or by using the DetectorMessenger in the run macro. Figure
708 11 shows a rendering of the 10 layers of the detector with the trajectories from a gamma event.

Listing 20: Absorber and Gap Volumes

```

709 // Absorber
710
711 absS = new G4Tubs("Abso",iRadius,oRadius,absThickness/2,startAngle,spanAngle);
712 absLV = new G4LogicalVolume(absS,absMaterial,"Absorber",0);
713 absPV = new G4PVPlacement(0,G4ThreeVector(0.0,0.0,-gapThickness/2),absLV,"Absorber",layerLV,
714 false,0,fCheckOverlaps);
715
716 // Gap

```

```

717 gapS = new G4Tubs("Gap", iRadius, oRadius, gapThickness/2, startAngle, spanAngle);
718 gapLV = new G4LogicalVolume(gapS, gapMaterial, "Gap", 0);
719 gapPV = new G4PVPlacement(0, G4ThreeVector(0.0, 0.0, absThickness/2), gapLV, "Gap", layerLV, false
720 , 0, fCheckOverlaps);
721

```

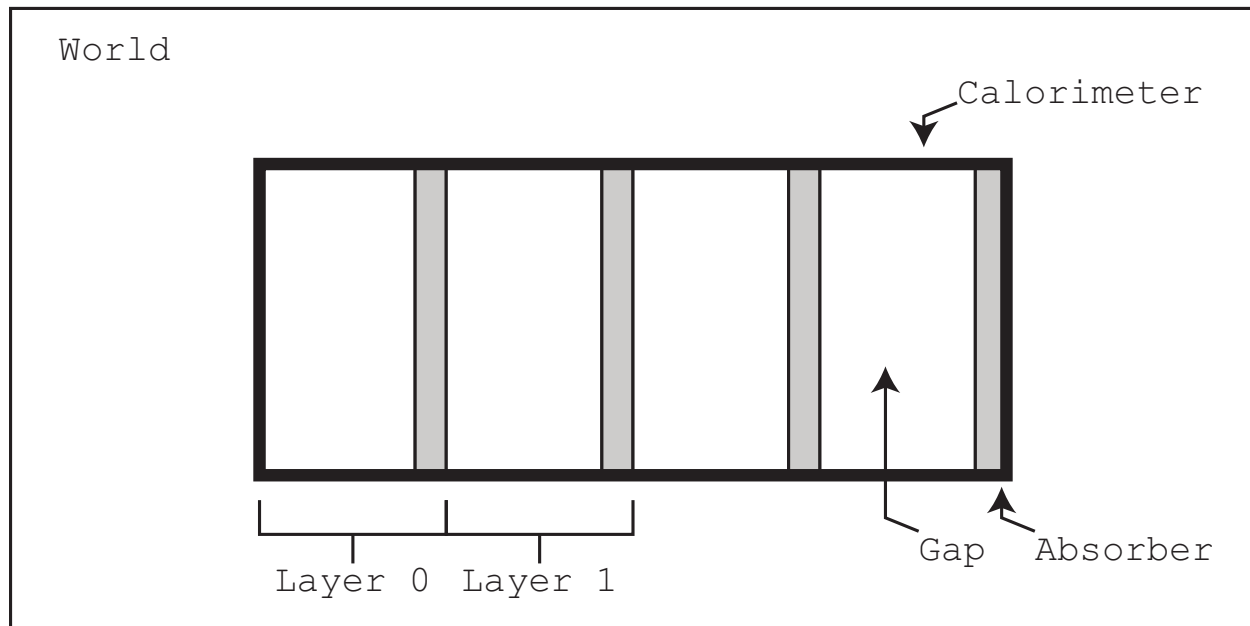


Fig. 10: World, Calorimeter, Layer and Absorber and Gap

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 combining 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 `EMStandardPhysics` 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].

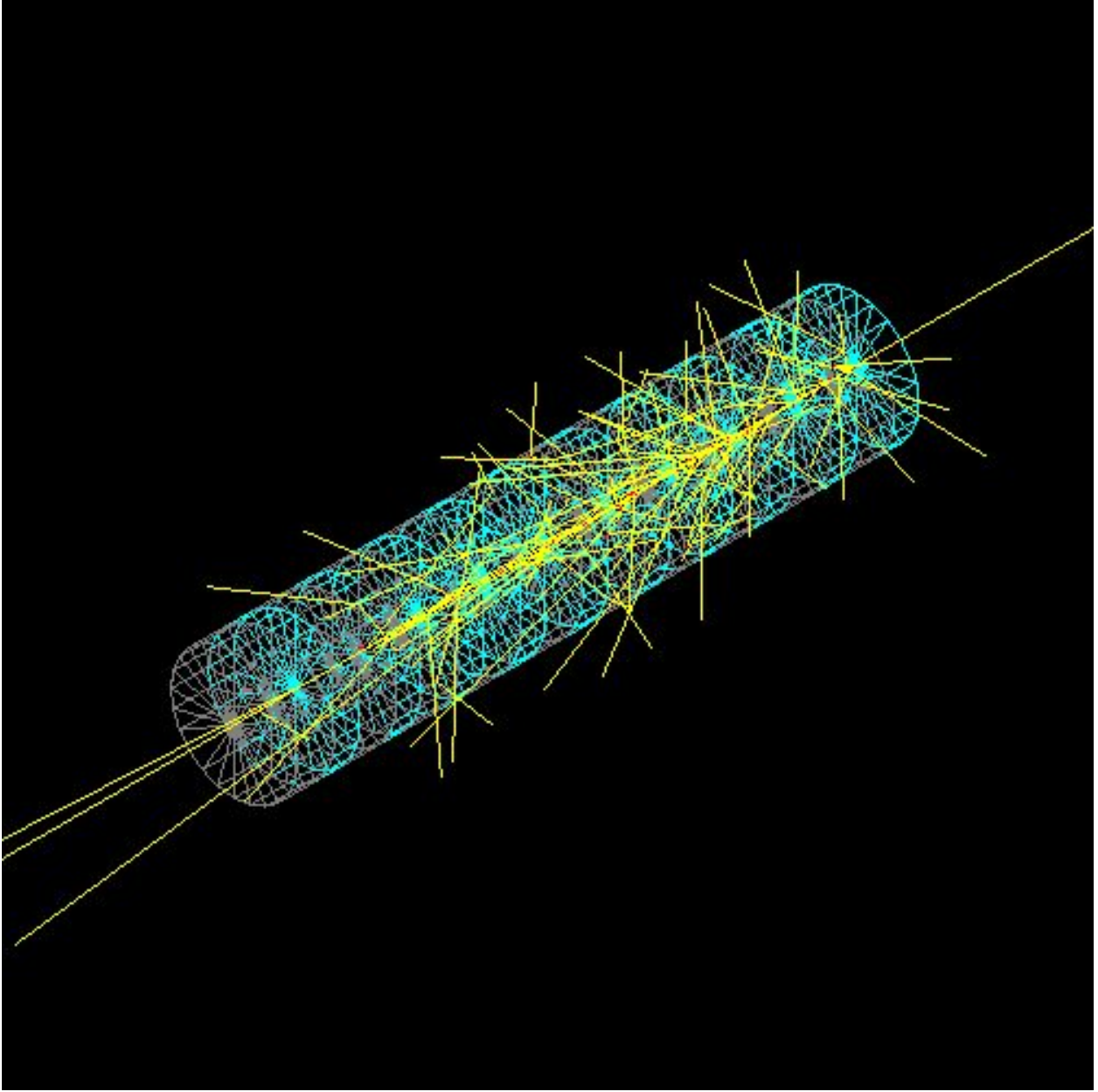


Fig. 11: 10 Layer Detector with a simulated gamma event

- `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 ${}^6\text{Li}(n, \alpha){}^3\text{H}$ interaction the `G4IonPhysics` list was used.

Listing 21: Implemented Physics List

```

744
745 1  /**
746     * PhysicsList
747 3  *
748     * Constructs the physics of the simulation
749 5  */
750 PhysicsList::PhysicsList() : G4VModularPhysicsList() {
751 7     currentDefaultCut    = 10*nm;
752
753 9     // Adding Physics List
754     //RegisterPhysics( new G4EmDNAPhysics());
755 11    RegisterPhysics( new G4EmStandardPhysics());
756     RegisterPhysics( new G4EmLivermorePhysics());
757 13    RegisterPhysics( new HadronPhysicsQGSP_BERT_HP());
758     RegisterPhysics( new G4IonPhysics());
759 15 }
760

```

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

Listing 22: Implemented Physics List

```

762
763 1 void PhysicsList::SetCuts() {
764     SetDefaultCutValue(10*nm);
765 3 }
766

```

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

Listing 23: Primary Event Generator

```

771
772 PrimaryGeneratorAction::PrimaryGeneratorAction() : G4VUserPrimaryGeneratorAction(), fParticleGun
773     (0) {
774 2     G4int nofParticles = 1;
775     fParticleGun = new G4ParticleGun(nofParticles);
776 4
777     // default particle kinematic
778 6     G4ParticleDefinition* particleDefinition = G4ParticleTable::GetParticleTable()->FindParticle("e
779         -");
780     fParticleGun->SetParticlePosition(G4ThreeVector(0.,0.,0.0));
781 8     fParticleGun->SetParticleDefinition(particleDefinition);
782     fParticleGun->SetParticleMomentumDirection(G4ThreeVector(0.,0.,1.));
783 10    fParticleGun->SetParticleEnergy(50.*MeV);
784 }
785

```

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

Listing 24: Generate Primaries

```

788
789 void PrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
790 {
791     // This function is called at the begining of event
792
793     // In order to avoid dependence of PrimaryGeneratorAction
794     // on DetectorConstruction class we get world volume
795     // from G4LogicalVolumeStore
796     G4double worldZHalfLength = 0;
797     G4LogicalVolume* worlLV = G4LogicalVolumeStore::GetInstance()->GetVolume("World");
798     G4Box* worldBox = 0;
799     if ( worlLV) worldBox = dynamic_cast< G4Box*>(worlLV->GetSolid());
800     if ( worldBox ) {
801         worldZHalfLength = worldBox->GetZHalfLength();
802     }
803     else {
804         G4cerr << "World volume of box not found." << G4endl;
805         G4cerr << "Perhaps you have changed geometry." << G4endl;
806         G4cerr << "The gun will be place in the center." << G4endl;
807     }
808
809     // Set gun position
810     fParticleGun->SetParticlePosition(G4ThreeVector(0., 0., -worldZHalfLength+1*cm));
811     fParticleGun->GeneratePrimaryVertex(anEvent);
812 }
813

```

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

```

828
829 /**
830 2 * @brief - Hit: a snapshot of the physical interaction of a track in the sensitive region of a
831     detector
832     *
833 4 * Contains:
834     * - Particle Information (type and rank (primary, secondary, tertiary ...))
835 6 * - Position and time
836     * - momentum and kinetic energy
837 8 * - deposition in volume
838     * - geometric information
839 10 */
840 class CaloHit : public G4VHit {
841 12 public:
842     CaloHit(const G4int layer);
843 14 ~CaloHit();
844
845 16     inline void* operator new(size_t);
846     inline void operator delete(void*);
847 18     void Print();
848
849 20 private:
850     G4double edep;           /* Energy Deposited at the Hit */
851 22     G4ThreeVector pos;      /* Position of the hit */
852     G4double stepLength;     /* Step Length */
853 24     G4ThreeVector momentum; /* Momentum of the step */
854     G4double kEnergy;        /* Kinetic Energy of the particle */
855 26     G4int trackID;          /* Track ID */
856     G4int parentID;          /* Parent ID */
857 28     G4ParticleDefinition* particle; /* Particle Definition */
858     G4int particleRank;       /* Primary, Secondary, etc */
859 30     G4VPhysicalVolume* volume; /* Physical Volume */
860     G4int layerNumber;        /* Copy Number of Layer */
861 32
862     public:
863 34         // Setter and Getters
864 };
865 36
866 typedef G4THitsCollection<CaloHit> CaloHitsCollection;
867 38 extern G4Allocator<CaloHit> HitAllocator;
868
869 40 inline void* CaloHit::operator new(size_t){
870     void *aHit;
871 42     aHit = (void *) HitAllocator.MallocSingle();
872     return aHit;
873 44 }
874
875 46 inline void CaloHit::operator delete(void *aHit){

```

```

876 HitAllocator.FreeSingle((CaloHit*) aHit);
877 }
878

```

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

Listing 26: Sensitive Detector

```

881
882 /**
883  * ProcessHits
884  *
885  * Adds a hit to the sensitive detector, depending on the step
886  */
887 G4bool CaloSensitiveDetector::ProcessHits(G4Step* aStep,G4TouchableHistory*) {
888
889     G4double edep = aStep->GetTotalEnergyDeposit();
890     G4double stepLength = aStep->GetStepLength();
891
892     // Getting the copy number
893     G4TouchableHistory* touchable = (G4TouchableHistory*)
894         (aStep->GetPreStepPoint()->GetTouchable());
895     G4int layerIndex = touchable->GetReplicaNumber(1);
896
897     // Creating the hit
898     CaloHit* newHit = new CaloHit(layerIndex);
899     newHit->SetTrackID(aStep->GetTrack()->GetTrackID());
900     newHit->SetParentID(aStep->GetTrack()->GetParentID());
901     newHit->SetEdep(edep);
902     newHit->SetStepLength(stepLength);
903     newHit->SetPosition(aStep->GetPreStepPoint()->GetPosition());
904     newHit->SetLayerNumber(layerIndex);
905     newHit->SetMomentum(aStep->GetPreStepPoint()->GetMomentum());
906     newHit->SetKineticEnergy(aStep->GetPreStepPoint()->GetKineticEnergy());
907     newHit->SetParticle(aStep->GetTrack()->GetDefinition());
908     newHit->SetVolume(aStep->GetTrack()->GetVolume());
909
910     // Adding the hit to the collection
911     hitCollection->insert( newHit );
912
913     return true;
914 }
915

```

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

Listing 27: Creating Sensitive Detectors

```

921
922 1 /**
923     * SetSensitiveDetectors
924 3     *
925     * Setting the Sensitive Detectors of the Detector
926 5     */
927 void DetectorConstruction::SetSensitiveDetectors() {
928 7     G4SDManager* SDman = G4SDManager::GetSDMpointer();
929     absSD = new CaloSensitiveDetector("SD/AbsSD", "AbsHitCollection");
930 9     SDman->AddNewDetector(absSD);
931     absLV->SetSensitiveDetector(absSD);
932 11
933     gapSD = new CaloSensitiveDetector("SD/GapSD", "GapHitCollection");
934 13     SDman->AddNewDetector(gapSD);
935     gapLV->SetSensitiveDetector(gapSD);
936 15 }
937

```

938 C. Analysis

939 Analysis of hit collection was preformed with ROOT. Once again there are other options (notably OpenScientist)

940 but previous experience was why ROOT was selected as the base for the Analysis framework. A singleton class

941 was written for the analysis which processed the hit collections, assigning the various results to root histograms.

942 User action classes EventAction and RunAction are called at the beginning and end of each run and event,

943 respectively (Listing 28,29). These classes allowed for the analysis code to be independent of the simulation.

Listing 28: Event Action

```

944
945 1 EventAction::EventAction() : G4UserEventAction() {
946     // Nothing to be Done Here
947 3 }
948
949 5 /**
950     * BeginOfEventAction
951 7     *
952     * @param const G4Event* event - event to be processed
953 9     *
954     * At the begining of an event we want to clear all the event
955 11     * accumulation variables.
956     */
957 13 void EventAction::BeginOfEventAction(const G4Event* event) {
958     Analysis::GetInstance()->PrepareNewEvent(event);
959 15 }
960
961 17 /**
962     * EndOfEventAction
963 19     *
964     * @param const G4Event* event - event to be processed

```

```

96521 *
966 * At the end of an event we want to call analysis to process
96723 * this event, and record the useful information.
968 */
96925 void EventAction::EndOfEventAction(const G4Event* event) {
970     Analysis::GetInstance()->EndOfEvent(event);
97127 }
972

```

Listing 29: Run Action

```

973
974 RunAction::RunAction() : G4UserRunAction() { }
975
976 void RunAction::BeginOfRunAction(const G4Run* run) {
977     G4cout<<"Starting run: " << run->GetRunID()<< G4endl;
978     Analysis::GetInstance()->PrepareNewRun(run);
979 }
980
981 void RunAction::EndOfRunAction(const G4Run* aRun) {
982     Analysis::GetInstance()->EndOfRun(aRun);
983 }
984

```

985 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.

$$E_{\text{dep,event}} = \sum E_{\text{dep,hit}} \quad (5)$$

986 ProcessHitCollection is called at the end of each event (Listing 30). Each hit is accessed and the layer at
 987 which it occurs is determined⁷. In addition the name of the volume is determined, and the energy deposition of the
 988 hit is added to the energy deposition of the event. If the hit occurred in the `absorber` layer and the particle is an
 989 electron the kinetic energy of that hit is also recorded.

Listing 30: Process Hit Collection

```

990
991 /**
992  * ProcessHitCollection
993  *
994  * @param G4VHitsCollection *hc
995  */
996 void Analysis::ProcessHitCollection(G4VHitsCollection *hc, G4int eventID) {
997
998     // Looping through the hit collection

```

⁷C 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 as an index into the histogram for the total of all layers in the material (either gap or absorber).

```

999 9      G4double hitColEdepTot_Abs[NUMLAYERS+1];    // Total EDep (abs) for Hit Collection
1000      G4double hitColEdepTot_Gap[NUMLAYERS+1];    // Total EDep (gap) for Hit Collection
100111     G4int PID;                                   // Parent ID
1002     for(int i= 0; i < NUMLAYERS+1; i++){
100313         hitColEdepTot_Abs[i] = 0.0;
1004         hitColEdepTot_Gap[i] = 0.0;
100515     }
1006
100717     // Energy Deposition of the event
1008     for(G4int i = 0; i < hc->GetSize(); i++){
100919         CaloHit* hit = (CaloHit*) hc->GetHit(i);
1010
101121         G4double eDep = hit->GetEdep();
1012         G4int layerNum = hit->GetLayerNumber();
101323         if (strcmp(hit->GetVolume()->GetName(),"Gap")){
1014             // Hit occurred in the Gap
101525             hitColEdepTot_Gap[layerNum] += eDep;
1016             (hHitTotEdepGap[layerNum])->Fill(eDep);
101727         }else if (strcmp(hit->GetVolume()->GetName(),"Absorber")){
1018             // Hit occurred in the Abs
101929             hitColEdepTot_Abs[layerNum] += eDep;
1020             (hHitTotEdepAbs[layerNum])->Fill(eDep);
102131
1022             /* Is this a secondary electron of the event? */
102333             if(hit->GetParticle()->GetPDGEncoding() == 11){
1024                 PID = hit->GetParentID();
102535                 if (PID < NUMPID){
1026                     (hSecElecKinAbs[layerNum][PID])->Fill(hit->GetKineticEnergy());
102737                 }
1028             }
102939         }
1030         else{
103141             G4cout<<"ERROR - Unkown Volume for sensitive detector"<<G4endl;
1032         }
103343     }
1034
103545     // Adding this Hit collection's energy deposited to event total
1036     for (int i = 0; i < NUMLAYERS; i++){
103747         // Incrementing each individual bin
1038         eventEdepTot_Abs[i] += hitColEdepTot_Abs[i];
103949         eventEdepTot_Gap[i] += hitColEdepTot_Gap[i];
1040
104151         // Last bin is Calorimter Total (all Abs layers and all Gap layers)
1042         eventEdepTot_Abs[NUMLAYERS] += hitColEdepTot_Abs[i];
104353         eventEdepTot_Gap[NUMLAYERS] += hitColEdepTot_Gap[i];
1044     }
104555 }
1046

```

1047 Finally, a run macro was written to control the entire run (Listing 31). The material and thickness of the detector
 1048 are declared (made possible by the use of `DetectorMessenger`), and then the detector is dynamically updated.
 1049 A ^{60}Co source is simulated by shooting photons of the 1.1732 MeV and 1.3325 MeV. The source particle is then
 1050 changed to a neutron, and thermal (0.025 eV) neutrons are shot at the detector. The thickness of the absorber is
 1051 then increased, the geometry updated, and the entire process repeated. As these runs tend to take a large amount of
 1052 time, GEANT4 was parallelized for use with MPI to take advantage of the cluster computing power.

Listing 31: Run Macro

```

1053 #
1054 1 /tracking/verbose 0
1055
1056 3 #
1057 # Setting up the detector
1058
1059 5 #
1059 /PolymerTransport/det/setAbsMat PS_Detector
1060 7 /PolymerTransport/det/setGapMat G4_POLYSTYRENE
1061 /PolymerTransport/det/setGapThick 0.3175 cm
1062 9 #
1063 /PolymerTransport/det/setAbsThick 15 um
1064 11 /PolymerTransport/det/update
1065 # Cobalt 60
1066 13 /gun/particle gamma
1067 /gun/direction 0 0 1
1068 15 /gun/energy 1.1732 MeV
1069 /run/beamOn 500000000 # 500 Million
1070 17 /gun/energy 1.3325 MeV
1071 /run/beamOn 500000000 # 500 Million
1072 19 # Neutron
1073 /gun/particle neutron
1074 21 /gun/energy 0.025 eV
1075 /run/beamOn 1000000 # 1 Million
1076 23 #
1077 /PolymerTransport/det/setAbsThick 25 um
1078 25 /PolymerTransport/det/update
1079

```

VI. RESULTS

A. Energy Deposition

The energy deposition was calculated 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 film. 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 from ^{60}Co . These features are not visible on the measured spectra due to the poor energy resolution of these films. There is also physical evidence of a lack of a Compton edge on the thinner films, but the films greater than 150 μm thick show some feature around 0.2 MeV. Films thinner than 150 μm show 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 deposition 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 overshadowed by the thick samples when plotted. It is also interesting to note that it is possible to observe the comparative effects of the α and ^3H in the neutron energy deposition spectra. The triton has a much shorter range ($\sim 10 \mu\text{m}$ in PS [9]) than the α ($\sim 60 \mu\text{m}$) 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 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 μm the average energy deposition was above 50% of the total Q -value of the reaction, and by 200 μm this average energy deposited approaches 95% of the total 4.78 MeV.

B. Secondary Electron Energy Distribution

The distribution of secondary electrons from photon interactions are plotted in Figure 14. From these results it can be concluded that 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 distribution was also quite energetic (with a maximum energy corresponding to 0.55 MeV) but with a much large probability of having a collision that produces an electron with a much lower energy.

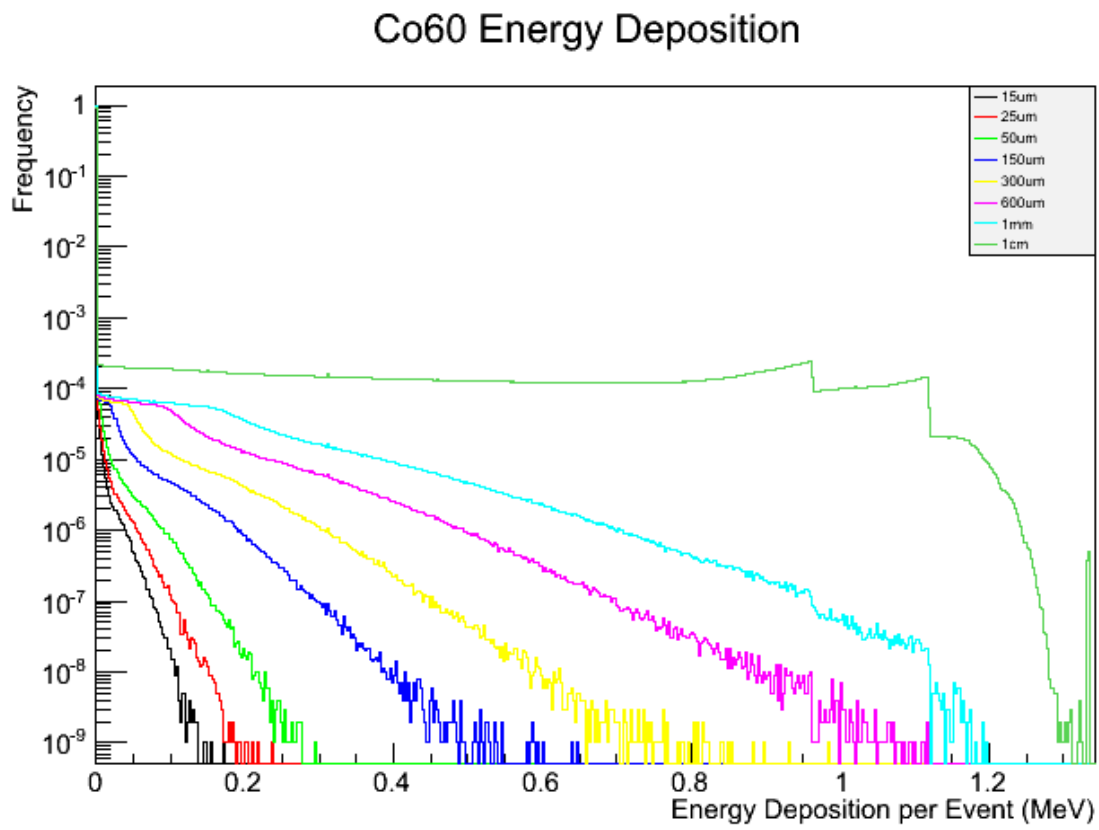


Fig. 12: Simulated Energy Depositon for a Single Film (gammas)

Energy Deposition

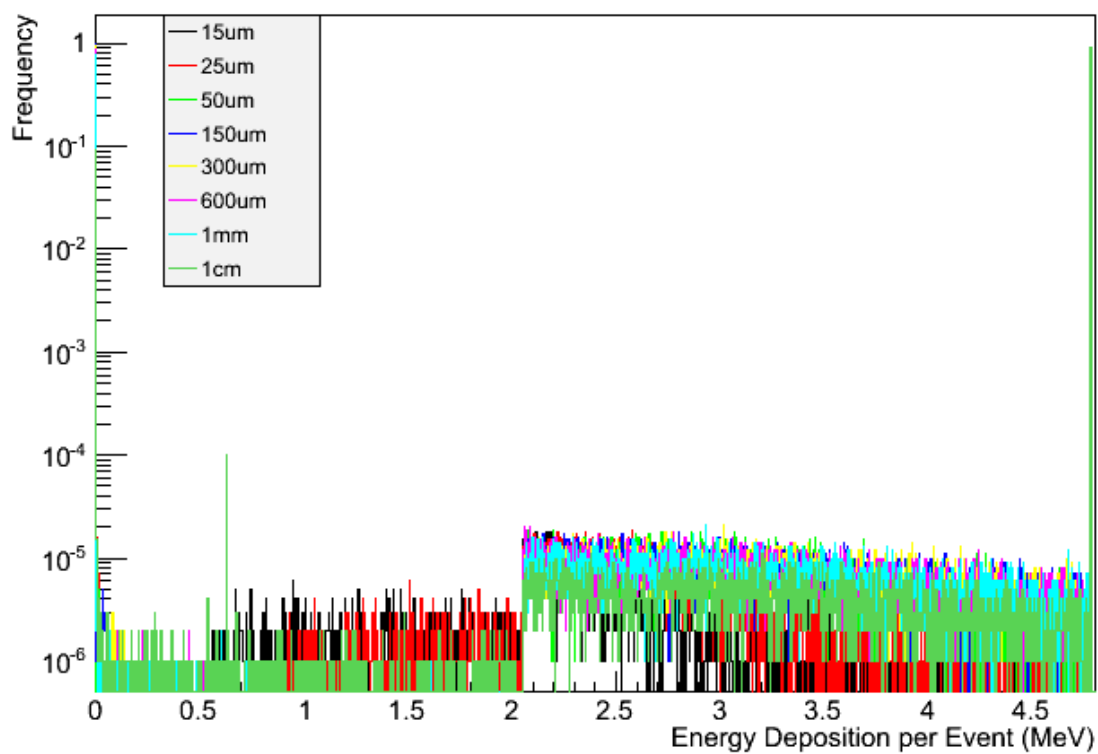
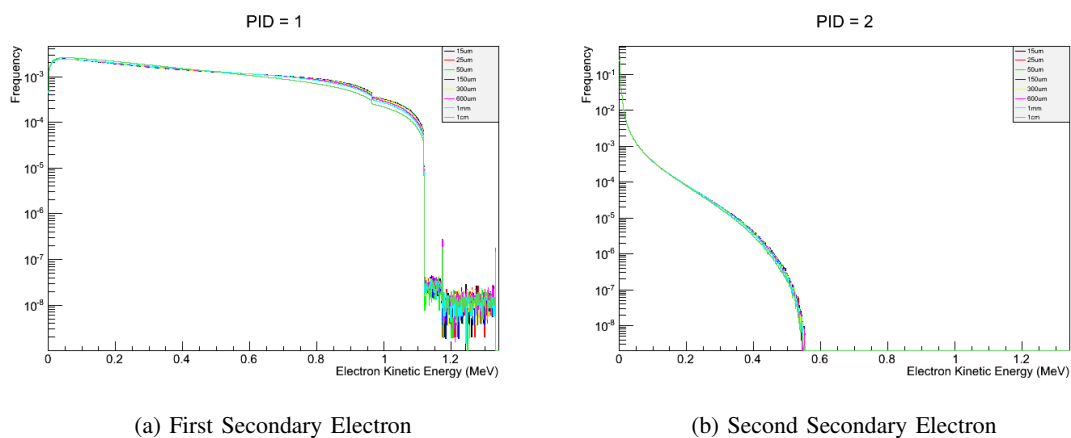


Fig. 13: Simulated Energy Deposition for a Single Film (neutrons)



(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 versatile implementation 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 reproducing 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 $15\mu\text{m}$ to $600\mu\text{m}$.

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 neutrons tend to deposit over 50% of their energy in the material for a $15\mu\text{m}$ film, and increasing to 96% for a 1 cm thick film. Figure 15 shows the average energy deposition as a function of thickness for neutrons and gammas, along with the calculated channel number (according to Equation 4). At thickness of less than $200\mu\text{m}$ there is significant separation between the average energy deposited by neutron events compared to gamma events. As the thickness of the films increased the average neutron energy approached the asymptotic limit of 4.78 MeV, while the average gamma energy increased. This creates less separation between the two, and provides less of an ability for neutron-gamma discrimination based on pulse height.

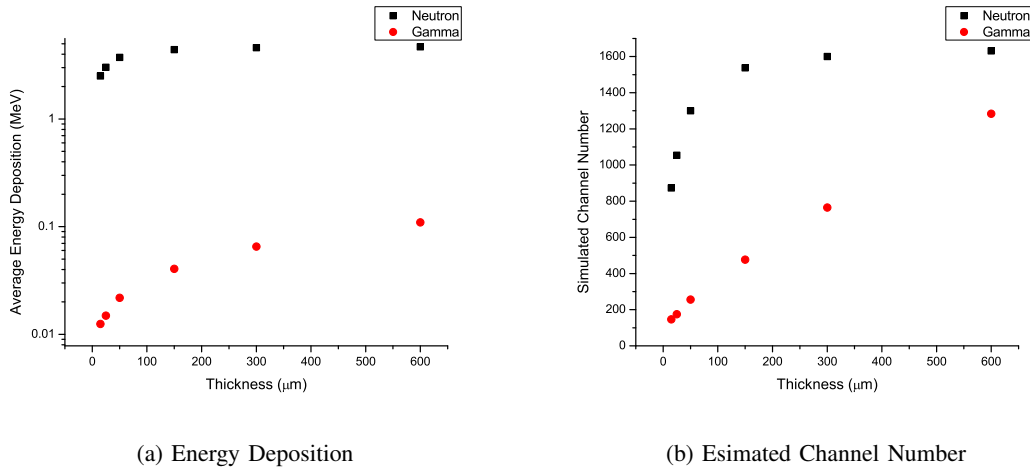


Fig. 15: Comparison between average neutron and gamma energy deposition

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, J. Apostolakis, H. Araujo, P. Arce, M. Asai, D. Axen, S. Banerjee, G. Barrand, F. Behner, L. Bellagamba, J. Boudreau, L. Broglia, A. Brunengo, H. Burkhardt, S. Chauvie, J. Chuma, R. Chytrcek, G. Cooperman, G. Cosmo, P. Degtyarenko, A. Dell'Acqua, G. Depaola, D. Dietrich, R. Enami, A. Feliciello, C. Ferguson, H. Fesefeldt, G. Folger, F. Foppiano, A. Forti, S. Garelli, S. Giani, R. Giannitrapani, D. Gibin, J. Gmez Cadenas, I. Gonzlez, G. Gracia Abril, G. Greeniaus, W. Greiner, V. Grichine, A. Grossheim, S. Guatelli, P. Gumplinger, R. Hamatsu, K. Hashimoto, H. Hasui, A. Heikkinen, A. Howard, V. Ivanchenko, A. Johnson, F. Jones, J. Kallenbach, N. Kanaya, M. Kawabata, Y. Kawabata, M. Kawaguti, S. Kelner, P. Kent, A. Kimura, T. Kodama, R. Kokoulin, M. Kossov, H. Kurashige, E. Lamanna, T. Lampn, V. Lara, V. Lefebvre, F. Lei, M. Liendl, W. Lockman, F. Longo, S. Magni, M. Maire, E. Medernach, K. Minamimoto, P. Mora de Freitas, Y. Morita, K. Murakami, M. Nagamatsu, R. Nartallo, P. Nieminen, T. Nishimura, K. Ohtsubo, M. Okamura, S. O'Neale, Y. Oohata, K. Paech, J. Perl, A. Pfeiffer, M. Pia, F. Ranjard, A. Rybin, S. Sadilov, E. Di Salvo, G. Santin, T. Sasaki, N. Savvas, Y. Sawada, S. Scherer, S. Sei, V. Sirotenko, D. Smith, N. Starkov, H. Stoecker, J. Sulkimo, M. Takahata, S. Tanaka, E. Tcherniaev, E. Safai Tehrani, M. Tropeano, P. Truscott, H. Uno, L. Urban, P. Urban, M. Verderi, A. Walkden, W. Wander, H. Weber, J. Wellisch, T. Wenaus, D. Williams, D. Wright, T. Yamada, H. Yoshida, and D. Zschesche, "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, H. Araujo, P. Dubois, M. Asai, G. Barrand, R. Capra, S. Chauvie, R. Chytrcek, G. Cirrone, G. Cooperman, G. Cosmo, G. Cuttone, G. Daquino, M. Donszelmann, M. Dressel, G. Folger, F. Foppiano, J. Generowicz, V. Grichine, S. Guatelli, P. Gumplinger, A. Heikkinen, I. Hrivnacova, A. Howard, S. Incerti, V. Ivanchenko, T. Johnson, F. Jones, T. Koi, R. Kokoulin, M. Kossov, H. Kurashige, V. Lara, S. Larsson, F. Lei, O. Link, F. Longo, M. Maire, A. Mantero, B. Mascialino, I. McLaren, P. Lorenzo, K. Minamimoto, K. Murakami, P. Nieminen, L. Pandola, S. Parlati, L. Peralta, J. Perl, A. Pfeiffer, M. Pia, A. Ribon, P. Rodrigues, G. Russo, S. Sadilov, G. Santin, T. Sasaki, D. Smith, N. Starkov, S. Tanaka, E. Tcherniaev, B. Tome, A. Trindade, P. Truscott, L. Urban, M. Verderi, A. Walkden, J. Wellisch, D. Williams, D. Wright, and H. Yoshida, "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 definition 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 ^{18}F produced by the $^{16}\text{O}(t,n)^{18}\text{F}$ reaction in neutron-irradiated lithium compounds containing oxygen," *The Journal of Chemical Physics*, vol. 72, no. 5, p. 3049, 1980.