

Geant4 for EIC luminosity monitor

Jaroslav Adam

BNL

BNL, January 29, 2020

EIC Software Tutorial

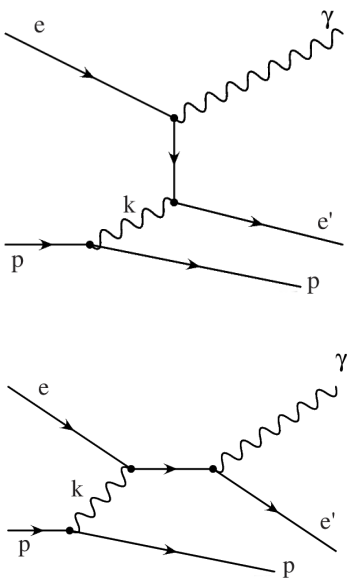
Introduction

- These slides give an overview of a standalone Geant4 project to simulate luminosity monitor for the EIC
- All the codes are in the following github repository:

<https://github.com/adamjaro/lmon>

- References to individual parts will be given throughout the presentation
- The only prerequisites to start working with a standalone Geant4 are basic knowledge of *C++* and *CMake*
- The EIC luminosity monitor here will serve as an illustration of Geant4 principles and functionalities
- In the following two tutorials this standalone Geant4 for luminosity monitor will be used to show how a standalone simulation can be integrated into existing detector full simulations

Mechanism of luminosity measurement at the EIC



- Luminosity is measured via elastic bremsstrahlung off electrons
- Independent of proton (nucleus) internal structure, large cross section $\sim \text{mb}$
- Luminosity monitor detects bremsstrahlung photons

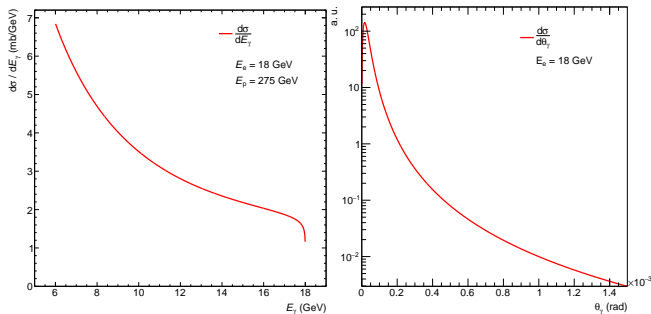


Figure: Bremsstrahlung cross section as a function of photon energy E_γ and polar angle θ_γ

Detector concept for luminosity measurement

- Following example of similar detector at ZEUS, HERA
- High luminosity demands two separate methods to count the bremsstrahlung photons:
 1. Photon conversion to e^+e^- pairs for precise DIS cross sections
 2. Direct, non converted photons for instantaneous collider performance

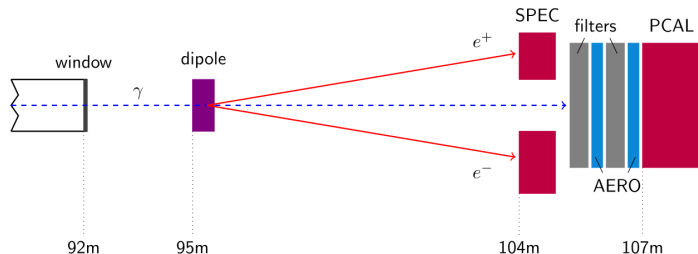


Figure: Layout of ZEUS luminosity detector

- Pairs are detected in spectrometer SPEC, direct photons in photon calorimeter PCAL

[Nucl.Instrum.Meth. A744 \(2014\) 80-90](#), [Nucl.Instrum.Meth. A565 \(2006\) 572-588](#)

Basic Geant4 principles

- Calculates passage of ionizing radiation in matter
- Can simulate light collection of optical scintillation and Cerenkov photons
- Geant4 is implemented as a set of *C++* classes, a project is built with *CMake*
- Three main steps of user interaction
 1. Generation of primary events
 2. Definition of detector volumes and construction materials
 3. Evaluating energy losses, optical photons and writing the output
- Geant4 runs in *steps* inside each volume along each particle trajectory
- At every step it calculates energy deposition and possible creation of secondary particles
- Geant4 can execute a user-defined function along every step
- Such a function is handed all information about the step, the volume, and particles involved in the step

Resources for Geant4

- Here is the list biased by my experience:
1. Book For Application Developers, [BookForApplicationDevelopers.pdf](#)
 - Comprehensive review of principles and functionalities
 - No need to read everything before doing something; first few pages of each chapter give enough information to begin with
 2. Class reference manual, <https://geant4.kek.jp/Reference/10.05.p01/classes.html>
 - Good to see class methods all in one place
 3. Example codes in Geant4 source directory
 4. Topical papers on special subjects, like light collection: [J.Phys.Conf.Ser. 798 \(2017\) no.1, 012218](#) or [Nucl.Instrum.Meth. A898 \(2018\) 30-39](#)
 5. Main page with references is here: <http://geant4.web.cern.ch/>

Generating events, single particle

```
8 //Geant headers
9 #include "G4ParticleGun.hh"
10 #include "G4ParticleTable.hh"
11 #include "G4ParticleDefinition.hh"
12 #include "G4SystemOfUnits.hh"
13
14 //local headers
15 #include "GeneratorAction.h"
16
17 //-----
18 GeneratorAction::GeneratorAction() : G4VUserPrimaryGeneratorAction(), fGun(0) {
19
20     fGun = new G4ParticleGun(1); //number of particles
21
22     G4String nam = "gamma";
23     G4ParticleDefinition *particle = G4ParticleTable::GetParticleTable()->FindParticle(nam);
24     fGun->SetParticleDefinition(particle);
25     fGun->SetParticleMomentumDirection(G4ThreeVector(0, 0, -1));
26     fGun->SetParticlePosition(G4ThreeVector(0, 0.2*cm, 0));
27     fGun->SetParticleEnergy(4.5*GeV);
28 } //GeneratorAction
29
30
31 //-----
32 void GeneratorAction::GeneratePrimaries(G4Event *evt) {
33
34     fGun->GeneratePrimaryVertex(evt);
35
36 } //GeneratePrimaries
```

- The generator class is derived from *G4VUserPrimaryGeneratorAction*
- It creates its *G4ParticleGun*
- Functions of the particle gun allow to set vertex position, direction and energy
- Units of *cm* and *GeV* are provided by Geant4
- Function *GeneratePrimaries* is called automatically by Geant4 at the beginning of each event

Generating events, uniform distribution

```
17 //-----
18 UniformGen::UniformGen() : G4VUserPrimaryGeneratorAction(), fGun(0) {
19
20     //minimal and maximal energy, in GeV
21     fEmin = 0.5;
22     fEmax = 20.;
23
24     fGun = new G4ParticleGun(1); //number of particles
25
26     G4String nam = "gamma";
27     G4ParticleDefinition *particle = G4ParticleTable::GetParticleTable()->FindParticle(nam);
28     fGun->SetParticleDefinition(particle);
29     fGun->SetParticleMomentumDirection(G4ThreeVector(0, 0, -1));
30
31     //uniform random generator for energy values
32     fRand = new CLHEP::HepRandom();
33
34 } //UniformGen
35
36 //-----
37 void UniformGen::GeneratePrimaries(G4Event *evt) {
38
39     G4double en = 0;
40     while(en < fEmin) {
41         en = fEmax * fRand->flat();
42     }
43     fGun->SetParticleEnergy(en*GeV);
44
45     fGun->GeneratePrimaryVertex(evt);
46
47 } //GeneratePrimaries
```

- Works like single particle generator from previous page
- Defines energy range over which the particles will be generated
- Energy for each event is obtained using *CLHEP::HepRandom*
- The *HepRandom::flat()* function provides values of uniform distribution

Generating physics events by reading a generator output

```
93 //particle loop
94 for(int itrk=0; itrk<ntrk; itrk++) {
95     getline(fIn, line);
96
97     //split the particle line
98     tokenizer< char_separator<char> > trkline(line, sep);
99     tokenizer< char_separator<char> >::iterator trk_it = trkline.begin();
100
101     //get the momentum
102     for(int i=0; i<2; i++) ++trk_it;
103     ss.str("");
104     ss.clear();
105     ss << *(trk_it++) << " " << *(trk_it++) << " " << *(trk_it++);
106     ss >> pz >> py >> px;
107
108     //get pdg
109     for(int i=0; i<3; i++) ++trk_it;
110     ss.str("");
111     ss.clear();
112     ss << *(trk_it++);
113     int pdg;
114     ss >> pdg;
115
116     //select the photon
117     if(pdg == 22) break;
118
119 } //particle loop
120
121 //generate the photon
122 G4ParticleGun gun(fGammaDef);
123
124 gun.SetParticleMomentum(G4ParticleMomentum(px*GeV, py*GeV, pz*GeV));
125 gun.SetParticlePosition(G4ThreeVector(vx*cm, vy*cm, vz*cm));
126
127 gun.GeneratePrimaryVertex(evt);
```

- Standard C++ functions are used to read a particular output of event generator, in TX format in this case
- Values of v_x , v_y and v_z and p_x , p_y and p_z are interpreted as *cm* and *GeV*
- They give vertex position and momentum of bremsstrahlung photon
- The event itself is again generated with *G4ParticleGun*

<https://github.com/adamjaro/lmon/blob/master/src/TxReader.cxx>

Creating a volume

```
29 //
30 ExitWindowV2::ExitWindowV2(const G4String& nam, G4double zpos, G4LogicalVolume *top):
31     Detector(), G4VSensitiveDetector(nam), fName(nam) {
32
33     G4cout << "ExitWindowV2: " << fName << G4endl;
34
35     G4double dz = 2.5*meter; // length along z
36     G4double radius = 10*cm; // inner radius
37     G4double thickness = 1*mm; // exit window thickness
38
39     //cylindrical U-shape
40     G4Tubs *shape = new G4Tubs(fName, radius, radius+thickness, dz/2., 90*deg, 180*deg);
41
42     //exit window material
43     G4Material *mat = G4NistManager::Instance()->FindOrBuildMaterial("G4_A1");
44
45     //logical volume
46     G4LogicalVolume *vol = new G4LogicalVolume(shape, mat, fName);
47
48     //visualization
49     G4VisAttributes *vis = new G4VisAttributes();
50     vis->SetColor(0, 1, 0, 0.5);
51     vis->SetForceSolid(true);
52     vol->SetVisAttributes(vis);
53
54     //100 mrad in x-z plane by rotation along y
55     G4RotationMatrix rot(G4ThreeVector(0, 1, 0), -0.1*rad); //typedef to CLHEP::HepRotation
56
57     //placement with rotation at a given position along z
58     G4ThreeVector pos(0, 0, zpos);
59     G4Transform3D transform(rot, pos); // is HepGeom::Transform3D
60
61     new G4PVPlacement(transform, vol, fName, top, false, 0);
62
63 } //ExitWindowV2
```

- The first step is a *shape*, *G4Tubs* in this case
- Shape with material makes *logical* volume
- Logical volume placed with *G4PVPlacement* makes *physical* volume
- The world of a given project is a composition of physical volumes

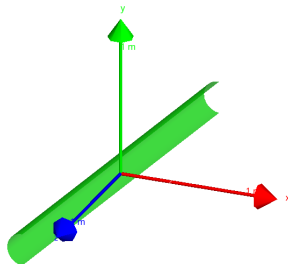


Figure: Outcome of the code on the left, half-pipe tilted along vertical axis y

Making a sensitive detector

```
9 #include "G4VSensitiveDetector.hh"
10
11 class ExitWindowV2 : public Detector, public G4VSensitiveDetector {
12
13     public:
14
15     ExitWindowV2(const G4String& nam, G4double zpos, G4LogicalVolume *top);
16
17     //Detector
18     virtual const G4String& GetName() const {return fName;}
19     virtual void CreateOutput(TTree *tree);
20     virtual void ClearEvent();
21     virtual void FinishEvent();
22
23     //G4VSensitiveDetector
24     virtual G4bool ProcessHits(G4Step *step, G4TouchableHistory*);
25
26     private:
27
```

- The particular detector has to be derived from *G4VSensitiveDetector*
- Geant4 then runs its *ProcessHits* function when making steps inside its sensitive volume
- All information about the step and associated particles comes from *G4Step*
- Information about the volume is provided by *G4TouchableHistory*

Putting pieces together in project executable and by Geant4 Actions

```
37 //default run manager
38 G4RunManager *runManager = new G4RunManager;
39
40 //detector construction
41 runManager->SetUserInitialization(new DetectorConstruction);
42
43 //physics
44 FTFP_BERT *physicsList = new FTFP_BERT;
45 G4OpticalPhysics *opt = new G4OpticalPhysics();
46 physicsList->RegisterPhysics(opt);
47 runManager->SetUserInitialization(physicsList);
48
49 //action
50 runManager->SetUserInitialization(new ActionInitialization);
51
52 //visualization
53 G4VisExecutive *visManager = new G4VisExecutive;
54 visManager->Initialize();
55
56 //user interface manager
57 G4UImanager *UImanager = G4UImanager::GetUIpointer();
58
```

Figure: Project executable run.cxx setting DetectorConstruction, physics and ActionInitialization

- The simulation is started by running the executable of the project
- The executable runs construction of all volumes
- Then loads the list of physics processes, including optional optical physics
- And finally sets the user-defined actions

```
17 //-----
18 void ActionInitialization::Build() const {
19
20     SetUserAction(new GeneratorAction);
21
22     SetUserAction(new EventAction);
23     SetUserAction(new RunAction);
24
25 } //build
```

Figure: Action initialization to set the event generator and functions to be run at the beginning of each event and at the beginning of the run

<https://github.com/adamjaro/lmon/blob/master/run.cxx>

<https://github.com/adamjaro/lmon/blob/master/src/ActionInitialization.cxx>

Constructing a simple framework

- Every detector, or part of the detector, in the luminosity framework inherits from *Detector* and from *G4VSensitiveDetector*
- Both base classes automatically run functions to manage output creation and Geant4 stepping
- It is easy to replace a simple model with more sophisticated detector

```
12 //abstract base class for detectors
13
14 class Detector {
15
16 public:
17
18     virtual ~Detector() {}
19
20     //add Detector to all Detectors during detector construction
21     virtual void Add(std::vector<Detector*> *vec) {vec->push_back(this);}
22     //name of logical sensitive volume for Detector with sensitive volume
23     virtual const G4String& GetName() const = 0;
24     //output of the detector
25     virtual void CreateOutput(TTree*) {}
26
27     virtual void ClearEvent() {} // beginning of event
28     virtual void FinishEvent() {} // end of event
29
30};
```

Figure: Base class for the detector

```
160 //-----
161 void DetectorConstruction::ConstructSDandField() {
162
163     G4cout << "DetectorConstruction::ConstructSDandField" << G4endl;
164
165     //detector loop
166     std::vector<Detector*>::iterator i;
167     for(i = fDet->begin(); i != fDet->end(); ++i) {
168         Detector *det = *i;
169
170         G4VSensitiveDetector *sd = dynamic_cast<G4VSensitiveDetector*>(det);
171         if(!sd) continue;
172
173         //detector inherits also from G4VSensitiveDetector, add it to Geant
174         G4SDManager::GetSDMpointer()->AddNewDetector(sd);
175         SetSensitiveDetector(det->GetName(), sd);
176
177         G4cout << " " << det->GetName() << G4endl;
178     } //detector loop
179 } //ConstructSDandField
180
181 } //ConstructSDandField
```

Figure: Registering every sensitive detector with Geant4 to make the mechanism with *ProcessHits* function working

<https://github.com/adamjaro/lmon/blob/master/include/Detector.h>

<https://github.com/adamjaro/lmon/blob/master/src/DetectorConstruction.cxx>

Writing output to ROOT tree

- Every detector is handed a pointer to the output ROOT TTree
- The detector can define its branches pointing to single numerical variables (example on the left)
- Or the branch could be and ROOT class, like histogram or TClonesArray
- Or the detector can add itself as a branch (example on the right)

```
36 Double_t fEdep; // total energy deposited in optical photon detector
37 ULong64_t fNphot; // number of photons in event
38 ULong64_t fNscin; // scintillation photons
39 ULong64_t fNcerenkov; // Cerenkov photons
40
41 Double_t fTmin; // time of first detected photon
42 Double_t fTmax; // time of last detected photon
43 Double_t fTavg; // average time of all detected photons
44
```

Figure: ROOT variables in optical photon detector *OpDet* to be written as a single branches in ROOT TTree

```
78 //-----
79 void ExitWinZEUS::CreateOutput(TTree *tree) {
80     //add this detector to the tree
81
82     fAddr = this;
83     tree->Branch(fNam, &fAddr);
84
85 } //CreateOutput
86
```

Figure: A detector which adds itself as a branch to the ROOT TTree; the *fAddr* is pointer to the detector object itself

<https://github.com/adamjaro/lmon/blob/master/include/OpDet.h>
<https://github.com/adamjaro/lmon/blob/master/src/ExitWinZEUS.cxx>

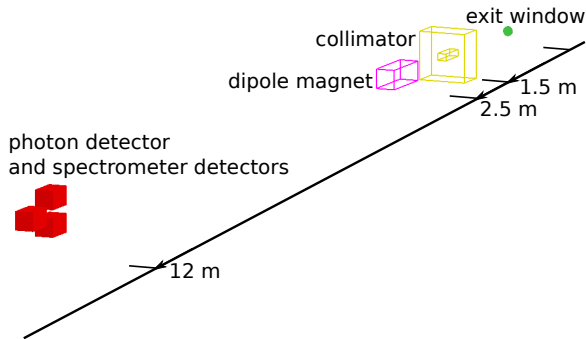
Running the simulation with a steering macro

```
2 #run macro for EIC nodes
3
4 #detectors and components, 1 = include, 0 = do not include
5 /lmon/construct/collim 1 # collimator
6 /lmon/construct/magnet 1 # spectrometer magnet
7 /lmon/construct/ewV2 1 # photon exit window
8 /lmon/construct/phot 1 # direct photon detector
9 /lmon/construct/up 1 # upper spectrometer calorimeter
10 /lmon/construct/down 1 # down spectrometer calorimeter
11
12 #init and run
13 /run/initialize
14
15 #input
16 /lmon/input/name /direct/eic+u/jadam/sim/lgen/data/lgen_18x275_10p1Mevt.tx
17
18 #output
19 /lmon/output/name ../data/lmon_18x275_all_0p25T_100kevt.root
20
21 #number of events
22 /run/beamOn 100000
23
```

- The project executable takes a steering macro as a command line parameter
- With the help of *G4GenericMessenger*, it is possible to configure the individual components
- Input from physics generator and output ROOT file also come from the messenger
- The messenger provides methods to create new commands

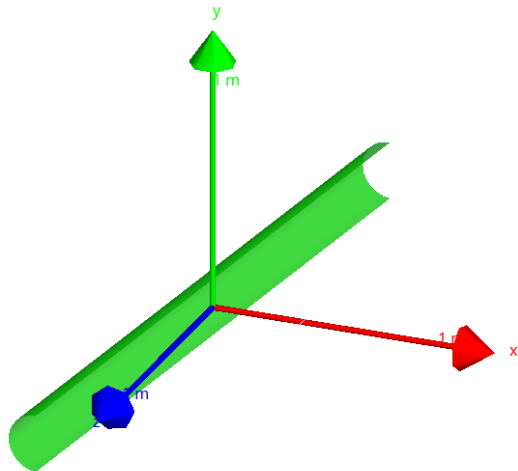
https://github.com/adamjaro/lmon/blob/master/run_eic.mac

Use case of luminosity monitor



- Full Geant4 model of all essential part of luminosity monitor
- Preliminary desing according to example from ZEUS
- Provides simulation chain from physics event generator to number of detected photoelectrons

Photon exit window



- Half-cylinder of 1 mm thick aluminum
- Conversion layer for the photons
- Also provides shielding against low energy background
- Although it will be a passive material, Geant4 can consider it as a detector to evaluate its performance as a conversion layer
- It was used as an example of sensitive detector in the previous section

<https://github.com/adamjaro/lmon/blob/master/src/ExitWindowV2.cxx>
<https://github.com/adamjaro/lmon/blob/master/include/ExitWindowV2.h>

Collimator

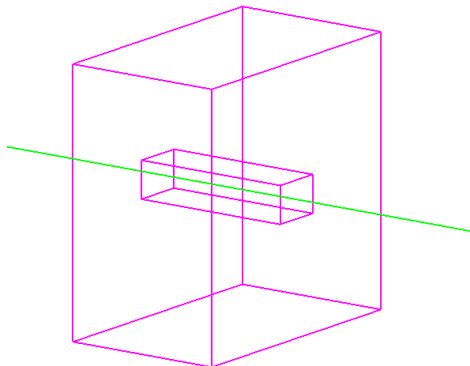


Figure: Bremsstrahlung photon passing through the collimator

- Block of stainless steel to shield the background
- The volume with opening inside is created with *G4SubtractionSolid*
- Outer shape and inner opening are created as *G4Box*

```
23 //inner aperture in x and y
24 G4double dx = 9.6*cm;
25 G4double dy = 7*cm;
26
27 G4double len = 30*cm; //length
28
29 G4double siz = 50*cm; //outer size
30
31 //collimator shape
32 G4String nam = "Collimator";
33 G4Box *outer = new G4Box(nam, siz/2, siz/2, len/2);
34 G4Box *inner = new G4Box(nam, dx/2, dy/2, len/2);
35 G4SubtractionSolid *shape = new G4SubtractionSolid(nam, outer, inner);
```

<https://github.com/adamjaro/lmon/blob/master/src/Collimator.cxx>
<https://github.com/adamjaro/lmon/blob/master/include/Collimator.h>

Spectrometer dipole magnet

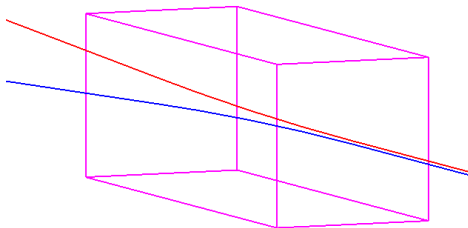


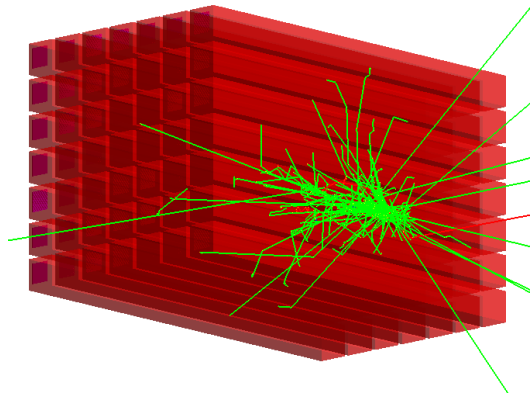
Figure: Electron and positron are deflected in the magnet

- The magnet in Geant4 is a volume with magnetic field
- The shape for the magnet is *G4Box*
- The magnetic field is *G4UniformMagField*
- The field is associated with the volume by *G4LogicalVolume::SetFieldManager*

```
26 //magnet shape
27 G4double dz = 60.*cm;
28 G4double xysiz = 10*cm;
29
30 G4String nam = "Magnet";
31 G4Box *mshape = new G4Box(nam, xysiz, xysiz, dz/2.);
32 G4LogicalVolume *mvol = new G4LogicalVolume(mshape, mat, nam);
33
34 //magnetic field
35 G4UniformMagField *field = new G4UniformMagField(G4ThreeVector(0.25*tesla, 0, 0));
36 G4FieldManager *fman = new G4FieldManager();
37 fman->SetDetectorField(field);
38 fman->CreateChordFinder(field);
39
40 mvol->SetFieldManager(fman, true);
```

<https://github.com/adamjaro/lmon/blob/master/src/Magnet.cxx>
<https://github.com/adamjaro/lmon/blob/master/include/Magnet.h>

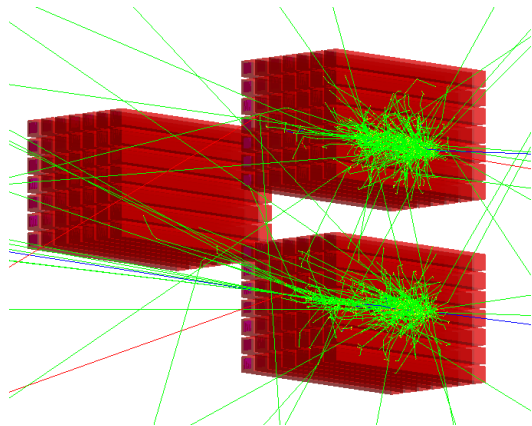
Photon detector



- Implemented as a composite calorimeter *CompCal*
- Consists of 7×7 PbWO_4 cells
- Each cell consists of 3×3 cm casing made of carbon fiber, 2 mm thick, holding the PbWO_4 crystal inside
- Length of each cell is 35 cm, same for casing and crystal
- Plot shows response to a 1 GeV photon
- The cell inherits from both *Detector* and *G4VSensitiveDetector*
- The calorimeter itself inherits only from *Detector*

<https://github.com/adamjaro/lmon/blob/master/src/CompCal.cxx>
<https://github.com/adamjaro/lmon/blob/master/include/CompCal.h>

Spectrometer detectors



- Pair of calorimeters for converted $e^+ e^-$ pairs
- Located in front of the photon detector
- Electrons and positrons are deflected by dipole magnet
- The plot shows event with e^+ and e^- at 3 GeV

<https://github.com/adamjaro/lmon/blob/master/src/CompCal.cxx>
<https://github.com/adamjaro/lmon/blob/master/include/CompCal.h>

Simulating scintillation and Cerenkov optical photons

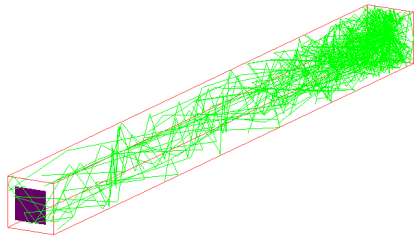


Figure: One calorimeter cell with 2 MeV deposition on the far side (facing the IP) and optical photon detector (magenta) on the opposite side. Optical photons are shown as green lines.

- Optical and scintillation properties for the Cell are defined in *OpTable*, Geant4 needs empiric parametrizations
- Optical photons are detected in optical detector *OpDet*, placed at the end of the cell
- The *ProcessHits* function of optical detector can get the origin of the photon

```
78 G4bool OpDet::ProcessHits(G4Step *step, G4TouchableHistory*) {
79
80 //total energy deposition in optical detector
81 fEdep += step->GetTotalEnergyDeposit();
82
83 //optical photons only since now
84 G4Track *track = step->GetTrack();
85 if(track->GetDynamicParticle()->GetParticleDefinition() != G4OpticalPhoton::OpticalPhotonDefinition()) {
86     return true;
87 }
88
89 //scintillation or Cerenkov photon
90 G4int ptype = track->GetCreatorProcess()->GetProcessType();
91 G4int pstype = track->GetCreatorProcess()->GetProcessSubType();
92 //scintillation photons
93 if(ptype == fScinType && pstype == fScinSubType) fNscin++;
94 //Cerenkov photons
95 if(ptype == fCerenkovType && pstype == fCerenkovSubType) fNcerenkov++;
```

<https://github.com/adamjaro/lmon/blob/master/src/Cell.cxx>

<https://github.com/adamjaro/lmon/blob/master/src/OpTable.cxx>

<https://github.com/adamjaro/lmon/blob/master/src/OpDet.cxx>

Summary

- With Geant4 it is straightforward to begin with simple things up to integrating into big projects

