Geant4 for EIC luminosity monitor

Jaroslav Adam

BNL

BNL, February 6, 2020

EIC Software Tutorial

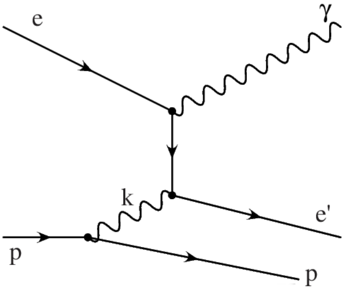
Introduction

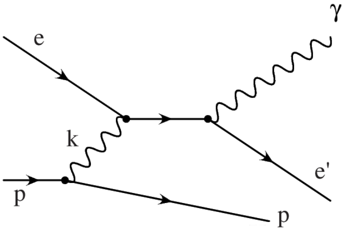
- These slides give an overview of a standalone Geant4 project to simulate luminosity monitor for the EIC
- All the codes are in the following github repository:

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 ∼mb
- Luminosity monitor detects bremsstrahlung photons

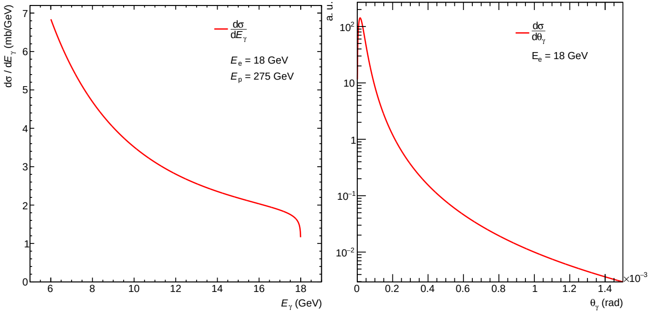


Figure: Bremsstrahlung cross section as a function of photon energy E_γ and polar angle θ_γ

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

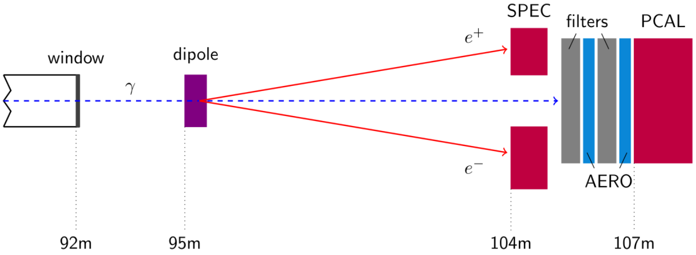


Figure: Layout of ZEUS luminosity detector

- 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

```
//Geant headers
#include "G4ParticleGun.hh"
#include "G4ParticleTable.hh"
#include "G4ParticleDefinition.hh"
#include "G4SystemOfUnits.hh"

//local headers
#include "GeneratorAction.h"

//_____________________________________________________________________________
GeneratorAction::GeneratorAction() : G4VUserPrimaryGeneratorAction(), fGun(0) {

  fGun = new G4ParticleGun(1); //number of particles

  G4String nam = "gamma";
  G4ParticleDefinition *particle = G4ParticleTable::GetParticleTable()->FindParticle(nam);
  fGun->SetParticleDefinition(particle);
  fGun->SetParticleMomentumDirection(G4ThreeVector(0, 0, -1));
  fGun->SetParticlePosition(G4ThreeVector(0, 0.2*cm, 0));
  fGun->SetParticleEnergy(4.5*GeV);

}//GeneratorAction

//_____________________________________________________________________________
void GeneratorAction::GeneratePrimaries(G4Event *evt) {

  fGun->GeneratePrimaryVertex(evt);

}//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

https://github.com/adamjaro/lmon/blob/master/src/GeneratorAction.cxx

Generating events, uniform distribution

```
//______________________________________________________________________________
UniformGen::UniformGen() : G4VUserPrimaryGeneratorAction(), fGun(0) {

  //minimal and maximal energy, in GeV
  fEmin = 0.5;
  fEmax = 20.;

  fGun = new G4ParticleGun(1); //number of particles

  G4String nam = "gamma";
  G4ParticleDefinition *particle = G4ParticleTable::GetParticleTable()->FindParticle(nam);
  fGun->SetParticleDefinition(particle);
  fGun->SetParticleMomentumDirection(G4ThreeVector(0, 0, -1));

  //uniform random generator for energy values
  fRand = new CLHEP::HepRandom();

}//UniformGen

//______________________________________________________________________________
void UniformGen::GeneratePrimaries(G4Event *evt) {

  G4double en = 0;
  while(en < fEmin) {
    en = fEmax * fRand->flat();
  }
  fGun->SetParticleEnergy(en*GeV);

  fGun->GeneratePrimaryVertex(evt);

}//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

https://github.com/adamjaro/lmon/blob/master/src/UniformGen.cxx

Generating physics events by reading a generator output

```
//particle loop
for(int itrk=0; itrk<ntrk; itrk++) {
  getline(fIn, line);

  //split the particle line
  tokenizer< char_separator<char> > trkline(line, sep);
  tokenizer< char_separator<char> >::iterator trk_it = trkline.begin();

  //get the momentum
  for(int i=0; i<2; i++) ++trk_it;
  ss.str("");
  ss.clear();
  ss << *(trk_it++) << " " << *(trk_it++) << " " << *(trk_it++);
  ss >> pz >> py >> px;

  //get pdg
  for(int i=0; i<3; i++) ++trk_it;
  ss.str("");
  ss.clear();
  ss << *(trk_it++);
  int pdg;
  ss >> pdg;

  //select the photon
  if(pdg == 22) break;

}//particle loop

//generate the photon
G4ParticleGun gun(fGammaDef);

gun.SetParticleMomentum(G4ParticleMomentum(px*GeV, py*GeV, pz*GeV));
gun.SetParticlePosition(G4ThreeVector(vx*cm, vy*cm, vz*cm));

gun.GeneratePrimaryVertex(evt);
```

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

- Standard C++ functions are used to read a particular output of event generator, in TX format in this case
- Values of *vx*, *vy* and *vz* and *px*, *py* and *pz* are interpreted as *cm* and *GeV*
- They give vertex position and momentum of bremsstrahlung photon
- The event itself is again generated with *G4ParticleGun*

Creating a volume

```
//____________________________________________________________________________
ExitWindowV2::ExitWindowV2(const G4String& nam, G4double zpos, G4LogicalVolume *top):
    Detector(), G4VSensitiveDetector(nam), fNam(nam) {

  G4cout << "ExitWindowV2: " << fNam << G4endl;

  G4double dz = 2.5*meter; // length along z
  G4double radius = 10*cm; // inner radius
  G4double thickness = 1*mm; // exit window thickness

  //cylindrical U-shape
  G4Tubs *shape = new G4Tubs(fNam, radius, radius+thickness, dz/2., 90*deg, 180*deg);

  //exit window material
  G4Material *mat = G4NistManager::Instance()->FindOrBuildMaterial("G4_Al");

  //logical volume
  G4LogicalVolume *vol = new G4LogicalVolume(shape, mat, fNam);

  //visualization
  G4VisAttributes *vis = new G4VisAttributes();
  vis->SetColor(0, 1, 0, 0.5);
  vis->SetForceSolid(true);
  vol->SetVisAttributes(vis);

  //100 mrad in x-z plane by rotation along y
  G4RotationMatrix rot(G4ThreeVector(0, 1, 0), -0.1*rad); //typedef to CLHEP::HepRotation

  //placement with rotation at a given position along z
  G4ThreeVector pos(0, 0, zpos);
  G4Transform3D transform(rot, pos); // is HepGeom::Transform3D

  new G4PVPlacement(transform, vol, fNam, top, false, 0);

}//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

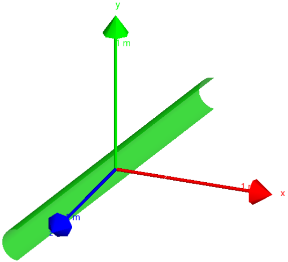


Figure: Outcome of the code on the left, half-pipe tilted along vertical axis *y*

https://github.com/adamjaro/lmon/blob/master/src/ExitWindowV2.cxx

Making a sensitive detector

```
#include "G4VSensitiveDetector.hh"

class ExitWindowV2 : public Detector, public G4VSensitiveDetector {

  public:

    ExitWindowV2(const G4String& nam, G4double zpos, G4LogicalVolume *top);

    //Detector
    virtual const G4String& GetName() const {return fNam;}
    virtual void CreateOutput(TTree *tree);
    virtual void ClearEvent();
    virtual void FinishEvent();

    //G4VSensitiveDetector
    virtual G4bool ProcessHits(G4Step *step, G4TouchableHistory*);

  private:
```

- 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*

https://github.com/adamjaro/lmon/blob/master/include/ExitWindowV2.h

Putting pieces together in project executable and by Geant4 Actions

```
//default run manager
G4RunManager *runManager = new G4RunManager;

//detector construction
runManager->SetUserInitialization(new DetectorConstruction);

//physics
FTFP_BERT *physicsList = new FTFP_BERT;
G4OpticalPhysics *opt = new G4OpticalPhysics();
physicsList->RegisterPhysics(opt);
runManager->SetUserInitialization(physicsList);

//action
runManager->SetUserInitialization(new ActionInitialization);

//visualization
G4VisExecutive *visManager = new G4VisExecutive;
visManager->Initialize();

//user interface manager
G4UImanager *UImanager = G4UImanager::GetUIpointer();
```

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

```
//_____________________________________________________________________________
void ActionInitialization::Build() const {

  SetUserAction(new GeneratorAction);

  SetUserAction(new EventAction);
  SetUserAction(new RunAction);

}//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

```
//abstract base class for detectors

class Detector {

  public:

    virtual ~Detector() {}

    //add Detector to all Detectors during detector construction
    virtual void Add(std::vector<Detector*> *vec) {vec->push_back(this);}
    //name of logical sensitive volume for Detector with sensitive volume
    virtual const G4String& GetName() const = 0;
    //output of the detector
    virtual void CreateOutput(TTree*) {}

    virtual void ClearEvent() {} // beginning of event
    virtual void FinishEvent() {} // end of event

};
```

Figure: Base class for the detector

```
//____________________________________________________________________________
void DetectorConstruction::ConstructSDandField() {

  G4cout << "DetectorConstruction::ConstructSDandField" << G4endl;

  //detector loop
  std::vector<Detector*>::iterator i;
  for(i = fDet->begin(); i != fDet->end(); ++i) {
    Detector *det = *i;

    G4VSensitiveDetector *sd = dynamic_cast<G4VSensitiveDetector*>(det);
    if(!sd) continue;

    //detector inherits also from G4VSensitiveDetector, add it to Geant

    G4SDManager::GetSDMpointer()->AddNewDetector(sd);
    SetSensitiveDetector(det->GetName(), sd);

    G4cout << "   " << det->GetName() << G4endl;
  }//detector loop

}//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)

```
  Double_t fEdep; // total energy deposited in optical photon detector
  ULong64_t fNphot; // number of photons in event
  ULong64_t fNscin; // scintillation photons
  ULong64_t fNcerenkov; // Cerenkov photons

  Double_t fTmin; // time of first detected photon
  Double_t fTmax; // time of last detected photon
  Double_t fTavg; // average time of all detected photons
```

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

```
//_____________________________________________________________________________
void ExitWinZEUS::CreateOutput(TTree *tree) {

  //add this detector to the tree

  fAddr = this;
  tree->Branch(fNam, &fAddr);

}//CreateOutput
```

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

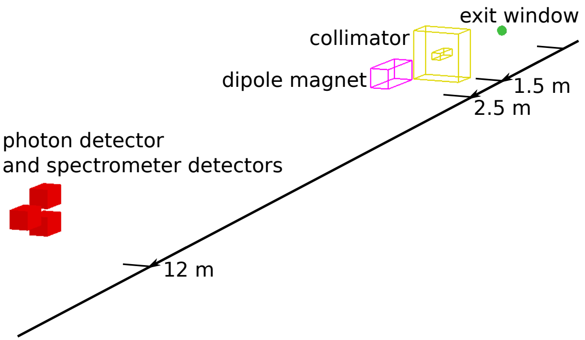
Running the simulation with a steering macro

```
#run macro for EIC nodes

#detectors and components, 1 = include, 0 = do not include
/lmon/construct/collim 1  # collimator
/lmon/construct/magnet 1  # spectrometer magnet
/lmon/construct/ewV2 1  # photon exit window
/lmon/construct/phot 1  # direct photon detector
/lmon/construct/up 1  # upper spectrometer calorimeter
/lmon/construct/down 1  # down spectrometer calorimeter

#init and run
/run/initialize

#input
/lmon/input/name /direct/eic+u/jadam/sim/lgen/data/lgen_18x275_10p1Mevt.tx

#output
/lmon/output/name ../data/lmon_18x275_all_0p25T_100kevt.root

#number of events
/run/beamOn 100000
```

- 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

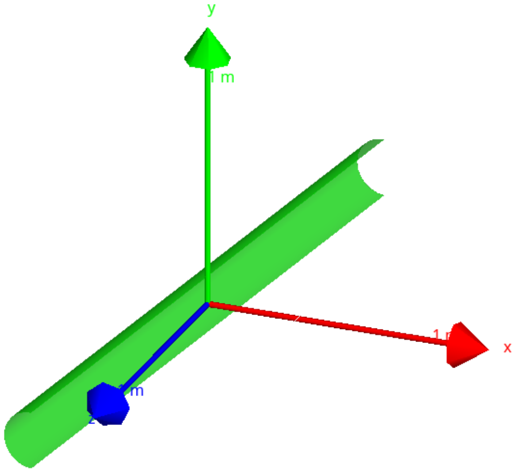
https://github.com/adamjaro/lmon/blob/master/run_eic.mac

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

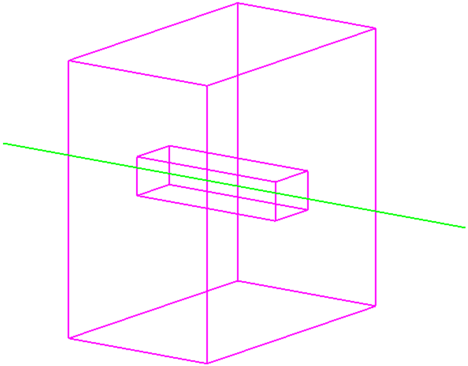


Figure: Bremsstrahlung photon passing through the collimator

- 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*

```
//inner aperture in x and y
G4double dx = 9.6*cm;
G4double dy = 7*cm;

G4double len = 30*cm; //length

G4double siz = 50*cm; //outer size

//collimator shape
G4String nam = "Collimator";
G4Box *outer = new G4Box(nam, siz/2, siz/2, len/2);
G4Box *inner = new G4Box(nam, dx/2, dy/2, len/2);
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

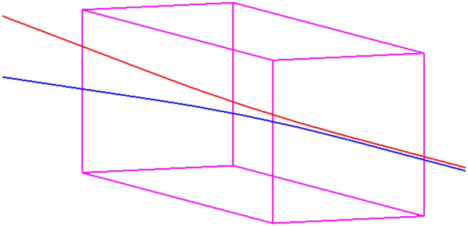


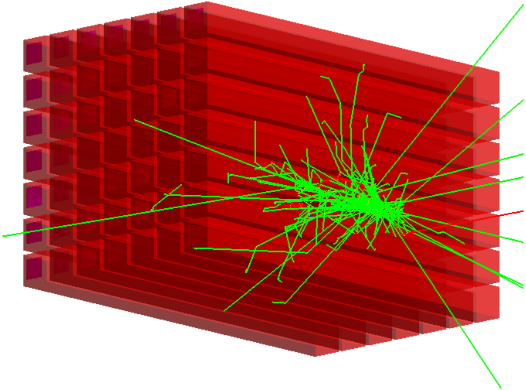
Figure: Electron and positron are deflected in the magnet

- 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*

```
//magnet shape
G4double dz = 60.*cm;
G4double xysiz = 10*cm;

G4String nam = "Magnet";
G4Box *mshape = new G4Box(nam, xysiz, xysiz, dz/2.);
G4LogicalVolume *mvol = new G4LogicalVolume(mshape, mat, nam);

//magnetic field
G4UniformMagField *field = new G4UniformMagField(G4ThreeVector(0.25*tesla, 0, 0));
G4FieldManager *fman = new G4FieldManager();
fman->SetDetectorField(field);
fman->CreateChordFinder(field);

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

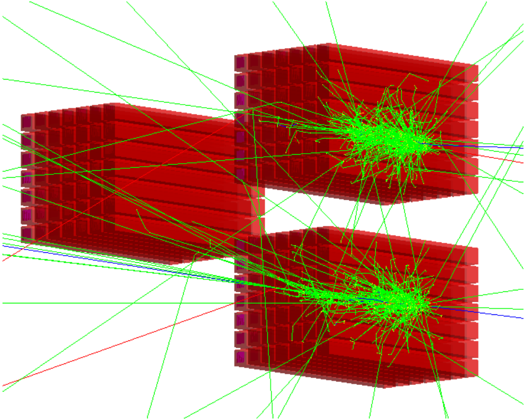


- Implemented as a composite calorimeter *CompCal*
- Consists of 7×7 $PbWO_4$ cells
- Each cell consists of 3×3 cm casing made of carbon fiber, 2 mm thick, holding the $PbWO_4$ crystal inside
- Length of each cell is 35 cm, same for casing and crystal
- Plot shows response to a 1 GeV photon
- The cell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

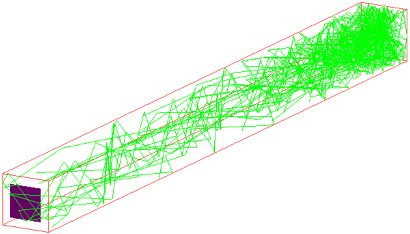


Figure: One calorimeter cell with 2 MeV deposition on the far side (facing the IP) and optical photon detector (magenta) on the opposite side. Optical photons are shown as green lines.

- 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

```
G4bool OpDet::ProcessHits(G4Step *step, G4TouchableHistory*) {

  //total energy deposition in optical detector
  fEdep += step->GetTotalEnergyDeposit();

  //optical photons only since now
  G4Track *track = step->GetTrack();
  if(track->GetDynamicParticle()->GetParticleDefinition() != G4OpticalPhoton::OpticalPhotonDefinition()) {
    return true;
  }

  //scintillation or Cerenkov photon
  G4int ptype = track->GetCreatorProcess()->GetProcessType();
  G4int pstype = track->GetCreatorProcess()->GetProcessSubType();
  //scintillation photons
  if(ptype == fScinType && pstype == fScinSubType) fNscin++;
  //Cerenkov photons
  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