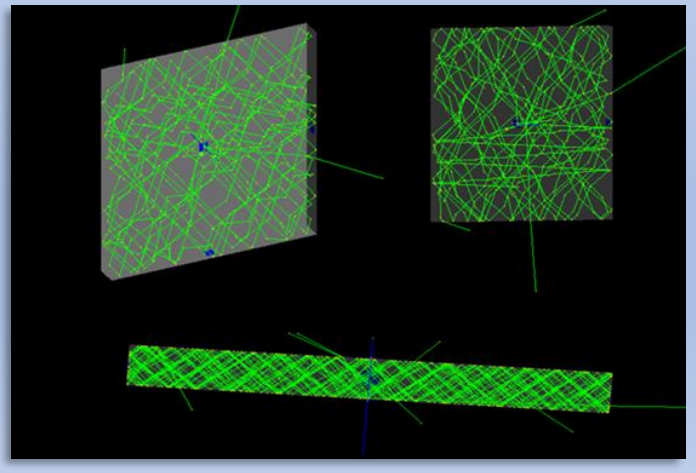
Simulation of optical photon propagation for generic scintillator-based detectors

Lecture 6

Sensitive Detector and Output file



Davide Serini
davide.serini@ba.infn.it

Hands-on schedule

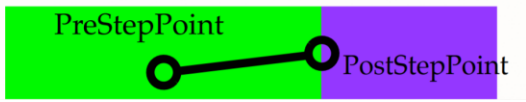
- **General G4 introduction**
  - **How to run the example B1**
    - ***Build and run the code***
    - ***Run a simple simulation using the User Interface***
    - ***Run a simple simulation using a macro***

- **Make your own project**
  - **Build your project:**
    - ***Main***
    - ***Geometry***
    - ***Physics list***
    - ***Write a macro***
  - **Build your simulation: data management tools:**
    - ***Sensitive Detector***
    - ***Hit collection***
    - ***Run Action***
    - ***Event Action***
  - **Make your simulation (final exercise)**

Simulation data storage: the «Sensitive Detector» concept

- **The STEP concept**

  A G4Step can be seen as a "**segment**" delimited by two points It contains "delta" information (energy loss along the step, time-offlight, etc)



- **The HIT concept**
  - A G4hit is **a snapshot of the physical interaction** of a track **in the sensitive region** of a detector.
  - In a "hit" you can store information associated with G4Step objects.
    - *the position and time of the step,*
    - *the momentum and energy of the track,*
    - *the energy deposition of the step,*
    - *geometrical information,*
- **The G4HitCollection:**
  - During the processing of a given event, represented by a G4Event object, **many objects of the hit class will be produced, collected and associated with the event**. Therefore, for each concrete hit class you must also prepare a concrete class derived from G4VHitsCollection, an abstract class which represents a vector collection of user defined hit.
- **The Sensitive Detector concept**
  - G4VSensitiveDetector is an abstract base class which **represents a detector.** The principal mandate of a sensitive detector **is the construction of hit** objects using information from steps along a particle track.
  - The ProcessHits() method of G4VSensitiveDetector performs this task using G4Step objects as input.
- **Sensitive Detector vs. Stepping Action:**
  - In the Sensitive Detector the hit is built during the processment of an event
  - In the Stepping Action the user process all the steps

Simulation data storage: Run Action, Process Hit and Event Action

- **Run Action:**
  - **Actions to do at the beginning and at the end of each «run» of simulations**
    - ***Creation of the input file and definition of the trees (begin of Run)***
    - ***Storage of the trees in the output file (end of Run)***
    - ***…***

- **Process Hit**
  - **Actions to do during the simulation in order to create and manage the hits in the sensitive detector**
    - ***Ouw definition of hit***
    - ***Step processing***
    - ***Hit storage***

- **Event Action**
  - **Actions to do at the beginning and at the end of each «event»:**
    - ***Creation of a collection of Hits associates to that event (begin of Event)***
    - ***Reading and storing the hits in the tree***

Exercise 2.A

- **Exercise 2.A:**
  - **Promote your detectors as «sensitive»**
  - **Create your Hit collection class and your definition of hit trying to retreive the following informations:**
    - ***The total energy deposited by all particles produced in each event («total dose»)***
    - ***The track length of one kind of particle (e.g. electrons) eventually produced inside your detector***
    - ***The amount of detected photon by your sensor (SiPM)***
  - **Processing the hit and:**
    - ***Print on screen (for debug mode) if and where those electrons are produced***
    - ***Store the hit at the EndOfEvent***
    - ***Clear the Sensitive detector status at the EndOfEvent***

For this exercise we need to act on the DetectorConstruction.cc OneHit.cc OneHit.hh SensitiveDetector.cc SensitiveDetector.hh

DetectorConstruction

```
//Sensitive Detector
  G4SDManager *SD_manager = G4SDManager::GetSDMpointer();
  G4String SDModuleName = "/SensitiveDetector";
  if(SD_manager->FindSensitiveDetector(SDModuleName,true))
    delete(SD_manager->FindSensitiveDetector(SDModuleName,true));
  SensitiveDetector *sensitiveModule = new SensitiveDetector(SDModuleName,"HitCollection");
  SD_manager->AddNewDetector(sensitiveModule);

  logicScint->SetSensitiveDetector(sensitiveModule);
  logicWrapping->SetSensitiveDetector(sensitiveModule);
  logicSiPM->SetSensitiveDetector(sensitiveModule);
```

Add here ALL your hit collections

Promote your logic volume to Sensitive

OneHit class

OneHit.hh

```
#ifndef OneHit_h
#define OneHit_h 1

//Include Native G4 Classes
#include "G4VHit.hh"
#include "G4Allocator.hh"
#include "G4THitsCollection.hh"
#include "G4ThreeVector.hh"
#include "G4UnitsTable.hh"
#include "G4Track.hh"

class OneHit : public G4VHit{
  public:

    OneHit();
    virtual ~OneHit();

    // methods from base class

    // set methods
    void SetEDep(G4double E)                          {fEDep = E;};
    void SetTrackLength(G4double L)                   {fTrackLength = L;};
    void SetPhotonCounter (G4int p)                   {fPhotonCounter = p;};

    // get methods
    G4double GetEDep()                          const { return fEDep;};
    G4double GetTrackLength()                   const { return fTrackLength;};
    G4int GetPhotonCounter()                    const { return fPhotonCounter;};

  private:
    G4double      fEDep;
    G4double      fTrackLength;
    G4int         fPhotonCounter;
  };

typedef G4THitsCollection<OneHit> HitCollection;


#endif
```

OneHit.cc

```
//Include Custom Classes
#include "OneHit.hh"

//Include Native G4 Classes
#include "G4UnitsTable.hh"
#include "G4VVisManager.hh"

#include <iomanip>

OneHit::OneHit():fEDep(0.), fTrackLength(0.), fPhotonCounter(0) {}

OneHit::~OneHit() {}
```

SensitiveDetector Class (header)

```
#ifndef SensitiveDetector_h
#define SensitiveDetector_h 1

//Include Custom Classes
#include "DetectorConstruction.hh"
#include "OneHit.hh"
#include "PhotonHit.hh"

//Include Native G4 Classes
#include "G4VSensitiveDetector.hh"
#include "G4TouchableHistory.hh"
#include "G4OpBoundaryProcess.hh"
#include <vector>

class G4Step;
class G4HCofThisEvent;

class SensitiveDetector : public G4VSensitiveDetector
{
public:
```

Mandatory methods

```
  //Constructor
  SensitiveDetector(const G4String &SDname, const G4String &HitCollectionName);
  //Distructor
  virtual ~SensitiveDetector();

  void Initialize(G4HCofThisEvent *HCE);
  G4bool ProcessHits(G4Step *step, G4TouchableHistory *ROhist);
  void EndOfEvent(G4HCofThisEvent *HCE);
```

Your function definitions to build the hit

```
  // Add methods
  // // Total deposit Energy
  void AddEdep(G4double edep) {fE += edep;};
  void DeleteTotalE()  { fE = 0.;};
  G4double GetTotalE()  const {return fE;};

  // // Total Track Length
  void AddTrackLength(G4double l) {fL += l;};
  void DeleteTotalTrackLength()  { fL = 0.;};
  G4double GetTotalTrackLength()  const {return fL;};

  // // Photon Counter
  void SetCounterStatus(G4int p) {fP +=p;};
  void ResetCounterStatus() {fP = 0;};
  G4int GetCounterStatus() const {return fP;};

  // // // // // //
  G4bool IsAnOpticalPhoton(G4Step* aStep);
  void CleanSDMemory();
  void PrintSDMemoryStatus();

private:
  HitCollection *fHitCollection;
  G4double      fE;
  G4double      fL;
  G4int         fP;
};

#endif
```

Your private members

Remember to add ALL your hit collection objects as member of the sensitive detector class and to include them

SensitiveDetector Class (source)

```
//Include Custom Classes
#include "SensitiveDetector.hh"
#include "OneHit.hh"
#include "PhysicsList.hh"

//Include Native G4 Classes
#include "G4ThreeVector.hh"
#include "G4SDManager.hh"
#include "G4ios.hh"
#include "G4TouchableHistoryHandle.hh"
#include "G4TouchableHistory.hh"
#include "Analysis.hh"
#include "G4RunManager.hh"
#include "G4UnitsTable.hh"
#include "G4Step.hh"
#include "G4HCofThisEvent.hh"
#include "G4Tubs.hh"
#include "G4OpticalPhoton.hh"
#include "G4OpBoundaryProcess.hh"
#include "G4GeometryTolerance.hh"
#include <math.h>

using namespace std;

static const G4double GeometricalTolerance = G4GeometryTolerance::GetInstance()->GetSurfaceTolerance();

SensitiveDetector::SensitiveDetector(const G4String &SDname,const G4String &HitCollectionName)
  : G4VSensitiveDetector(SDname),
    fHitCollection(NULL),
    fE(0.),
    fL(0.),
    fP(0),
{
  G4cout<<"Creating SD with name: "<<SDname<<G4endl;
  collectionName.insert(HitCollectionName);

}

SensitiveDetector::~SensitiveDetector(){}

void SensitiveDetector::Initialize(G4HCofThisEvent *HCE)
{
  fHitCollection = new HitCollection(GetName(),collectionName[0]);
  static G4int HCID = G4SDManager::GetSDMpointer()->GetCollectionID(collectionName[0]); //<<-- this is to get an ID for the colletionName[0]
  G4cout<<"*** "<<fHitCollection->GetName()<<" initialized [ID = "<<HCID<<"]"<<G4endl;
  HCE->AddHitsCollection(HCID, fHitCollection);
}
```

Class constructor: Remember to initialize all the members (also all the hit collections)

Class destructor

Add to your sensitive Detector the Hit collection and give it an univoque ID

SensitiveDetector Class (source)

```
G4bool SensitiveDetector::ProcessHits(G4Step *aStep, G4TouchableHistory *)
{
  G4double eDep   = aStep->GetTotalEnergyDeposit();
  G4double deltaL = aStep->GetStepLength();
  G4int counter = 0;

  if(SensitiveDetector::IsAnOpticalPhoton(aStep))
    counter ++;

  //SensitiveDetector::SetCounterStatus(counter);
  SensitiveDetector::AddEdep(eDep);
  G4String myParticle = "e-";
  G4String thisParticle = aStep->GetTrack()->GetParticleDefinition()->GetParticleName();
  G4String thisVolume   = aStep->GetTrack()->GetVolume()->GetName();
  if(thisParticle==myParticle){
    //G4cout<<"[DEBUG]: I Found "<<thisParticle<<G4endl;
    //G4cout<<"[DEBUG]: I Found "<<thisParticle<<" in the Volume: "<<thisVolume<<G4endl;
    SensitiveDetector::AddTrackLength(deltaL);
  }
  G4int pDetected = 0;
  if(thisParticle=="opticalphoton"){
    G4String procName = aStep->GetPostStepPoint()->GetProcessDefinedStep()->GetProcessName();
    G4bool checkAbsorption = procName.contains("Absorption");
    if(checkAbsorption==true and thisVolume=="SiPM")
      pDetected +=1;
    }

  SensitiveDetector::SetCounterStatus(pDetected);



  return true;
}
```

This is the main part in which you create your hit starting from the information that you can retreive by

- **G4Step**
- **G4Track**

SensitiveDetector Class (source)

Your «AUXILIARY» function definitions

```
////////////////////////////////////////////////////////////////////////////
G4bool SensitiveDetector::IsAnOpticalPhoton(G4Step* aStep){
  G4bool flag = false;
  G4Track *aTrack = aStep->GetTrack();
  const G4ParticleDefinition* aDef = aTrack->GetParticleDefinition();
  G4String aName = aDef->GetParticleName();
  if(aName=="opticalphoton"){
    flag=true;
  }
  return flag;
}

void SensitiveDetector::CleanSDMemory(){
  SensitiveDetector::DeleteTotalE();
  SensitiveDetector::DeleteTotalTrackLength();
  SensitiveDetector::ResetCounterStatus();
}

void SensitiveDetector::PrintSDMemoryStatus(){
  G4double TotE = SensitiveDetector::GetTotalE();
  G4double TotL = SensitiveDetector::GetTotalTrackLength();
  G4int TotP = SensitiveDetector::GetCounterStatus();
  G4cout<<" 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0"<<G4endl;
  G4cout<<" Total Energy deposited = "<<G4BestUnit(TotE,"Energy")<<G4endl;
  G4cout<<" Total TrackLength = "<<G4BestUnit(TotL,"Length")<<G4endl;
  G4cout<<" Total Photons = "<<TotP<<G4endl;
  G4cout<<" 0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0"<<G4endl;
```

Check if the particle is an Optical photon

Reset the SD Memory status

Get the SD Memory status

SensitiveDetector Class (source)

```
void SensitiveDetector::EndOfEvent(G4HCofThisEvent*)
{
  //Fill the hits
  OneHit *aHit = new OneHit();
  G4double TotE = SensitiveDetector::GetTotalE();
  G4double TotL = SensitiveDetector::GetTotalTrackLength();
  G4int TotP = SensitiveDetector::GetCounterStatus();

  aHit->SetEDep(TotE);
  aHit->SetTrackLength(TotL);
  aHit->SetPhotonCounter(TotP);

  fHitCollection->insert(aHit);

  SensitiveDetector::PrintSDMemoryStatus();
  G4cout<<"!!!EndOfEvent!!!"<<G4endl;
  SensitiveDetector::CleanSDMemory();
}
```

EndOfEvent:

- **Retreive the current SD memory status**
- **Insert your score in the hit**
- **Print the status on screen**
- **Print a message for the end of event**
- **Clear the memory**