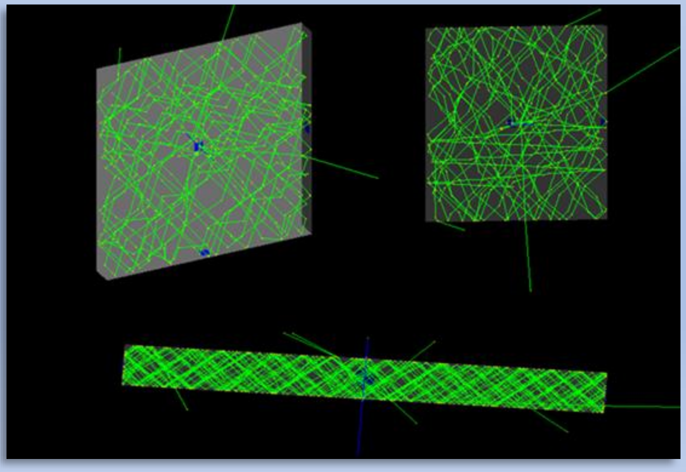
Simulation of optical photon propagation for generic scintillator-based detectors

Lecture 6

Sensitive Detector



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)**

Exercise 2.B

- **Exercise 2.B:**
  - **Create at the beginning of the run your root output in order to save the information stored**
  - **Retrieve the hit information stored at the end of each event and save them in the root file**
  - **Close the root file**

For this exercise we need to act on the RunAction.cc RunAction.hh EventAction.cc EventAction.hh

RunAction Class

```
#ifndef RunAction_h
#define RunAction_h 1

//Include Native G4 Classes
#include "G4UserRunAction.hh"
#include "G4Accumulable.hh"
#include "globals.hh"
#include "g4root.hh"

class G4Run;

class RunAction : public G4UserRunAction
{
  public:
    RunAction();
    virtual ~RunAction();

    virtual void BeginOfRunAction(const G4Run*);
    virtual void   EndOfRunAction(const G4Run*);

  private:

};

#endif
```

Mandatory methods

```
#include "DetectorConstruction.hh"

//Include Native G4 Classes
#include "G4RunManager.hh"
#include "G4Run.hh"
#include "G4AccumulableManager.hh"
#include "G4LogicalVolumeStore.hh"
#include "G4LogicalVolume.hh"
#include "G4UnitsTable.hh"
#include "G4SystemOfUnits.hh"
#include "G4GeneralParticleSource.hh"
#include "Analysis.hh"

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......

RunAction::RunAction()
: G4UserRunAction()
{
  G4RunManager::GetRunManager()->SetPrintProgress(1);
  auto analysisManager = G4AnalysisManager::Instance();
  G4cout << "Using " << analysisManager->GetType() << G4endl;

  //Create ntuple
  analysisManager->CreateNtuple("Primary_Hit", "Primary_Hit");
  analysisManager->CreateNtupleIColumn(0,"eventID");
  analysisManager->CreateNtupleDColumn(0,"EDep_Hit_MeV");
  analysisManager->CreateNtupleDColumn(0,"TrackLength_Hit_mm");
  analysisManager->CreateNtupleIColumn(0,"PhotonDetected");
  analysisManager->FinishNtuple(0);
}
```

Class constructor: Instance the analysisManager and create your Root Tree structure

```
//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......

RunAction::~RunAction()
{
  delete G4AnalysisManager::Instance();
}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......

void RunAction::BeginOfRunAction(const G4Run*)
{
  auto analysisManager = G4AnalysisManager::Instance();
  analysisManager->OpenFile("Results.root");
}
```

Call the AnalysisManager and create the root file

```
//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......

void RunAction::EndOfRunAction(const G4Run*)
{
  auto analysisManager = G4AnalysisManager::Instance();
  // save histograms & ntuple
  analysisManager->Write();
  analysisManager->CloseFile();

}
```

Call the AnalysisManager and write/close the root file

EventAction Class (header)

EventAction.hh

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

//Include Native G4 Classes
#include "G4UserEventAction.hh"
#include "globals.hh"

class RunAction;

/// Event action class
///

class EventAction : public G4UserEventAction
{
  public:
    EventAction();
    virtual ~EventAction();

    virtual void BeginOfEventAction(const G4Event* event);
    virtual void EndOfEventAction(const G4Event* event);

  private:
    HitCollection *GetHitsCollection(G4int HCID,const G4Event* event) const;
    G4int fHCID;

    PhotonHitCollection *GetPhotonHitsCollection(G4int PHCID,const G4Event* event) const;
    G4int fPHCID;
};

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......

#endif
```

Mandatory methods

Retrieve the hits collection

EventAction Class (source)

EventAction.cc

```
//Include Custom Classes
#include "EventAction.hh"
#include "RunAction.hh"

//Include Native G4 Classes
#include "G4Event.hh"
#include "G4RunManager.hh"
#include "G4SDManager.hh"
#include "G4SystemOfUnits.hh"

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......

EventAction::EventAction()
: G4UserEventAction(),
  fHCID(-1),

{}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......

EventAction::~EventAction()
{}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
HitCollection*
EventAction::GetHitsCollection(G4int HCID, const G4Event* event) const
{
  auto hitsCollection = static_cast<HitCollection*>(event->GetHCofThisEvent()->GetHC(HCID));

  if ( ! hitsCollection ) {
    G4ExceptionDescription msg;
    msg << "Cannot access hitsCollection ID " << HCID;
    G4Exception("B4cEventAction::GetHitsCollection()",
      "MyCode0003", FatalException, msg);
  }

  return hitsCollection;
```

Initialize the hits collection ID

Take your hits collection (or interrupt the simulation if it is not available)

EventAction Class: EndOfEvent

EventAction.cc

```
//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
void EventAction::BeginOfEventAction(const G4Event*)
{}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......

void EventAction::EndOfEventAction(const G4Event* event)
{
  G4int nEvt = event->GetEventID();

  auto analysisManager = G4AnalysisManager::Instance();
  if ( fHCID == -1 ) {
    fHCID = G4SDManager::GetSDMpointer()->GetCollectionID("HitCollection");
  }

  // ***Primary Hit Collection*** //
  auto HC = GetHitsCollection(fHCID, event);
  if(HC->entries()>0){
    G4double EnergyDep = 0.;
    G4double TrackLength = 0.;
    G4int PhotonDetected = 0;

    EnergyDep   = (*HC)[0]->GetEDep();
    TrackLength = (*HC)[0]->GetTrackLength();
    PhotonDetected = (*HC)[0]->GetPhotonCounter();

    analysisManager->FillNtupleIColumn(0,0,  nEvt);
    analysisManager->FillNtupleDColumn(0,1,  EnergyDep*MeV);
    analysisManager->FillNtupleDColumn(0,2,  TrackLength*mm);
    analysisManager->FillNtupleIColumn(0,3,  PhotonDetected);
    analysisManager->AddNtupleRow(0);
  }
```

Call your analysis manager

Find your hits collection

Read your hits collection

Save your information and begin a new Row

Read your results

Open the root file with the command:

`>> root -l <NAME_FILE.root>`

There are 2 simple ways to explore the root file:

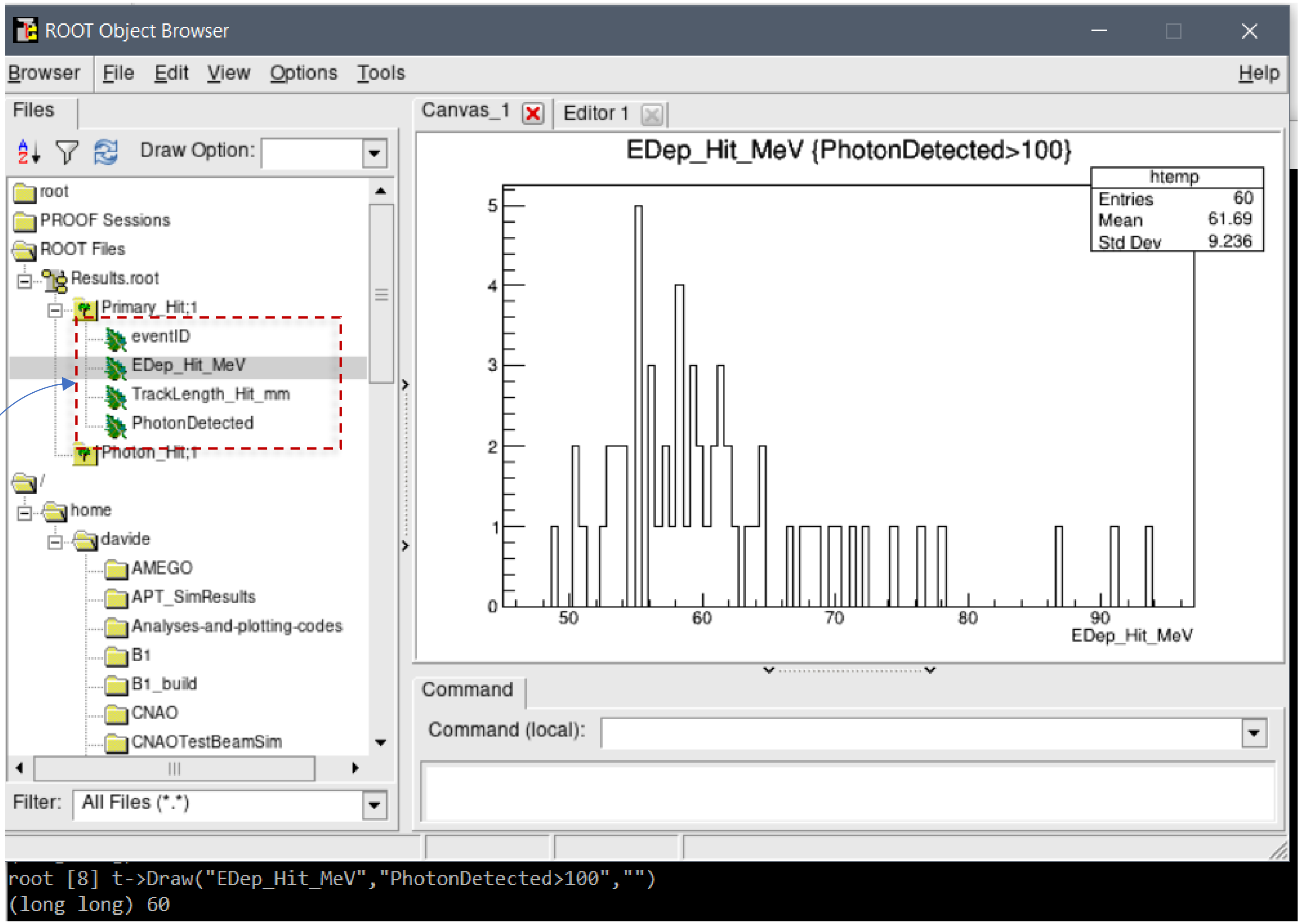
a) With the Graphical interface (TBrowser)

use the command:

`>> TBrowser a`

You can explore the file with the window menu

Select a variable (branch) to see an histogram that collect the values that variable.



Read your results

b) With command line:

- Explore your file with the command:

  `_file0->ls()`

  This command will show you all the trees, histograms, graps ecc.. Inside your file

- Load the tree that you want and open with the command:

  `>> Ttree* t = (Ttree*)_file0->Get(«Primary_Hit»)`

- Show each single entry with the command:

  `>> t->Show(numberOfEvent) (e.g. t->Show(0))`

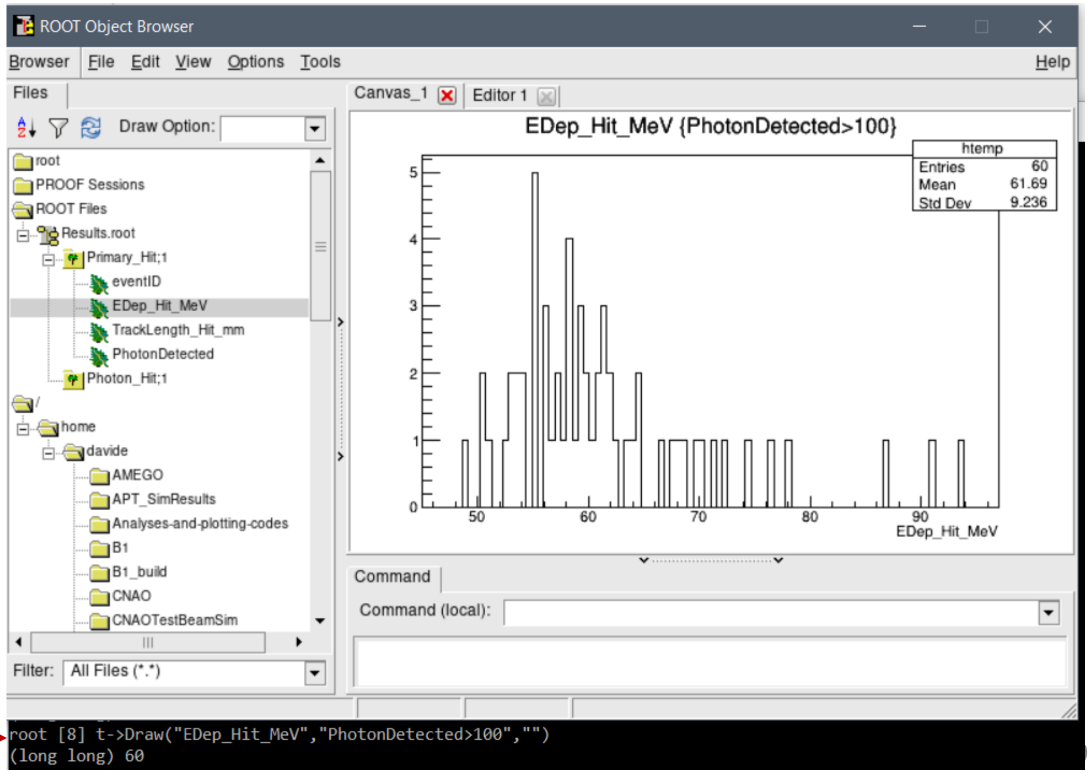
```
root [9] t->Show(0)
======> EVENT:0
 eventID           = 0
 EDep_Hit_MeV      = 52.3969
 TrackLength_Hit_mm = 3.78893
 PhotonDetected    = 916
```

- Make an histogram with the command:

>> t->Draw(«VariableName », «cutConditions», «GraphicalOptions»)

`t->Draw(«Edep_Hit_Mev», «photonDetected>100», «»)`

For more information about Root
https://root.cern/doc/master/index.html



RunActionMessenger

- **We need to define by macro the file name in order to perform different simulations at the same time.**
  - **We can use our custom messengers (like those used in the GPS)**
  - **We need to define our custom actions (commands) and initialize the Messenger in the RunAction**

RunActionMessenger.hh

```
#ifndef RunActionMessenger_h
#define RunActionMessenger_h 1

#include "globals.hh"
#include "G4UImessenger.hh"

class RunAction;

class G4UIdirectory;
class G4UIcmdWithAnInteger;
class G4UIcmdWithAString;
class G4UIcmdWithABool;

class RunActionMessenger : public G4UImessenger
{
public:

  RunActionMessenger(RunAction* );
  virtual ~RunActionMessenger();

  virtual void SetNewValue(G4UIcommand* ,G4String );

private:

  RunAction*                  fRunAction;
  G4UIcmdWithAString*         fSetNameOfOutputFile;
};
#endif
```

RunActionMessenger.cc

```
#include "RunActionMessenger.hh"

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
RunActionMessenger::RunActionMessenger(RunAction* runAction)
  : fRunAction (runAction)
{
  fSetNameOfOutputFile=
    new G4UIcmdWithAString("/RunManager/NameOfOutputFile",this);
  fSetNameOfOutputFile->SetGuidance("Name of Output File");
  fSetNameOfOutputFile->SetParameterName("Name of Output File",false);
  fSetNameOfOutputFile->AvailableForStates(G4State_PreInit,G4State_Idle);
  fSetNameOfOutputFile->SetToBeBroadcasted(false);
}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
RunActionMessenger::~RunActionMessenger() {
  delete fSetNameOfOutputFile;
}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......
void RunActionMessenger::SetNewValue(G4UIcommand* command,G4String newValue)
{
  if( command == fSetNameOfOutputFile)
    fRunAction->SetNameOfOutputFile(newValue);
}
```

Manager link to RunAction

RunAction.hh

```
#ifndef RunAction_h
#define RunAction_h 1

#include "G4UserRunAction.hh"
#include "G4Accumulable.hh"
#include "globals.hh"

class G4Run;
class RunActionMessenger;


class RunAction : public G4UserRunAction
{
  public:
    RunAction();
    virtual ~RunAction();

    // virtual G4Run* GenerateRun();
    virtual void BeginOfRunAction(const G4Run*);
    virtual void    EndOfRunAction(const G4Run*);

    void SetNameOfOutputFile(G4String name){fOutputFileName=name;};
    G4String GetNameOfOutputFile() const { return fOutputFileName;};

    void  GetInstance();

  private:
    RunActionMessenger *fRunMessenger;
    G4String fOutputFileName;
};

#endif
```

RunAction.cc

Run Action constructor ➔ We need to instantiate the RunMessenger to retrieve the fOutputFileName

```
RunAction::RunAction(): G4UserRunAction(), fOutputFileName("./Data")
{
  fRunMessenger = new RunActionMessenger(this);
```

Thanks to the link to the RunActionMessenger the fOutputFileName is correctly setted by the user through macro

```
/RunManager/NameOfOutputFile Prova2
```

```
void RunAction::BeginOfRunAction(const G4Run*)
{
  G4String fileName   = fOutputFileName;
  G4String fileOutput = fileName;
  fileOutput += ".root";
  G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();
  analysisManager->OpenFile(fileOutput);
}
```

Exercise 2.C (intermediate)

- **Exercise 2.C:**
  - **Try to create another hit collection called "PhotonHitCollection" in which you want to store the Wavelength and the VolumeID every time an optical photon is absorbed in a volume**
    - ***To save the volume use the following enumeration:***

```
G4int id = -1;

if(thisVolume.contains("Scintillator"))
  id = 0;
else if(thisVolume.contains("Wrapping"))
  id=1;
else if(thisVolume.contains("SiPM"))
  id=2;
else
  id = -1;
```

If we have more SiPMs, we need also to know which one collect the photon!

    - ***NB this time more photons will be produced during a single event so you need to store several hit per event (not only one time at the EndOfEvent)***
    - ***Suggestion:*** Create your "processOpticalPhoton" function, store your hit each time the function will be called (e.g. every time an optical photon is absorbed) and call it during the main process hit function

  - **Retrieve the hit information stored at the end of each event and save them in the root file**

Exercise 2.C (intermediate)

- **Exercise 2.C *<u>How to evaluate the wavelength</u>*:**

$$\lambda = \frac{hc}{E}$$

```
G4double h = CLHEP::h_Planck/((CLHEP::eV)*(CLHEP::ns));
G4double c = CLHEP::c_light/((CLHEP::nm)/(CLHEP::ns));

G4double e0 = aStep->GetPostStepPoint()->GetKineticEnergy();
G4double e1 = e0/CLHEP::eV;
G4double wl = c*h/e1;
```

In this way you will have the photon wavelength in nanometers

<u>Suggestions:</u>

A . Remember that you will create a new Hit collection object so:

1. You need to initialize properly it ***everywhere*** you call your collections (it is a new object with a proper name, ID…)
2. you need to include it ***everywhere*** you call your collections:
   - In the DetectorConstruction
   - In the Sensitive Detector
   - In the Event Action

B. Create a new tree called «Photon_Tree» in which there will be more entries for a single event.

You need a way to connect the «Primary_Tree» with the «Photon_Tree» for further analysis (in general).
A simple way to do that is to store the eventID in all trees that you will fill in your simulation.

Ex2.tar.gz

Links & References

Blank

wget 'https://istnazfisnucl-my.sharepoint.com/:u:/g/personal/serini_infn_it/EX-1XCKrJdlImqv1wKKZY20BiiMb9b5eAdP1ozxKD2wLyQ?e=Y7DJeZ&download=1' -O Blank.tar.gz

Ex0

wget 'https://istnazfisnucl-my.sharepoint.com/:u:/g/personal/serini_infn_it/EfRyYDwP7gZJruci7aTx1V4Bhpm-EaYmroRFsEnDIGLRTQ?e=PICleo&download=1' -O Ex0.tar.gz

Ex1

wget 'https://istnazfisnucl-my.sharepoint.com/:u:/g/personal/serini_infn_it/ERnXARL7WtJMhMlfPvs8AW8BukNkJhT4Sytw0Q7yY3r3zA?e=51DStU&download=1' -O Ex1.tar.gz

Ex2

wget 'https://istnazfisnucl-my.sharepoint.com/:u:/g/personal/serini_infn_it/EarCKaIe981CoYTsmdFJBvYBbg0W2RjnASt-y2fBQeJrZg?e=7ifD24&download=1' -O Ex2.tar.gz

A complete Reference of the Hits and Analysis is available in the Application Developers Guide, Chapter 4 and Chapter 9

Final exercise (advance)

- **Starting from the blank project write a simulation of a «toy-module» compose of:**
  - **A 10x10x2 cm^3 calorimeter of scintillating material**
  - **A plane of 1x1x10 cm^3 squared fibers along one module view**
  - **A readout SiPM strip to read the light collected at one fiber end**

- **Simulate mu+ particles cross the module in different position and with differen energies and readout the collected light by the fiber plane**