Scoring using Senstive Detector

Introduction

Now we know following:

(a) Creation of detector geometry

(b) Creation of Physics List

(c) Running Simulation

In this lecture we will learn about :

Extracting information from the simulation using Sensitive detector

Why Sensitive Detector ?

Geant4 does full simulation silently.

To get the meaningful information out, we need to interfer.

Can be done by writing option user hooks.

One such hook is writing a UserSteppingAction.

This gives us access to each and every step.

Is it really required ?? Not always.....

What if I wanted to get the information only if the particle is in a particular volume.

Here Sensitive detector comes into the picture.

Sensitive Detector

Act like a readout system of your detector

Allows user to focus on the desired volume, by making them sensitive to particles and their interactions.

No need to check for volume in which you are (as you already know)

Gives complete information like energy deposited, step size, position, timing information etc., everything what one can get from SteppingAction.

Allows user to accumulate all the hits within a sensitive volume.

Recipe and components to create a Sensitive detector based application

Create the detector geometry (done)

Create Physics List (done)

Create Primary Generator (done)

Create Sensitive Detector class

Create Hit Class

Create Hit Collection

Fill the hit collections in the Sensitive Detector

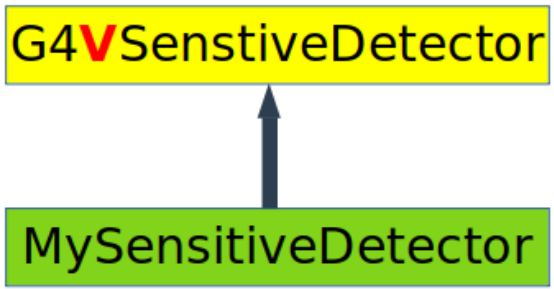
Register the hit collection from each sensitive detector to container of hit collection for the entire event

Create an object of Sensitive Detector class and register it with Sensitive detector manager

Declare the required logical volume as Sensitive by attaching it to the object of Sensitive detector class

Steps to create a Sensitive Detector

1. Write a class to implement your Sensitive Detector
2. Attach an object of this class to your Logical Volume
3. Register this object with Sensitive Detector Manager



Sensitive Detector interface : G4VSensitiveDetector

Inherit your Sensitive Detector class from G4VSensitiveDetector

Important point

Once a logical volume is made sensitive, all its placement will become sensitive.

To distinguish between different placements use copy number or the name assigned to physicsl volume

It not necessary to have a different SD class for different Logical Volume.

Different logical volume can share same SD object.

More than one SD objects can be made from same SD class, but each object should have different name.

Implementing a Sensitive Detector class

Inherit the G4VSensitiveDetector class

Add the desire optional function, useful for logging and debugging information

Implement the mandatory function, ProcessHits

ProcessHits function is called for each step within you sensitive volume.

```
#include "G4VSensitiveDetector.hh"
#include "G4HCofThisEvent.hh"

//Public Inheritance
class NaI_SD : public G4VSensitiveDetector {
public:
    //Constructor
    NaI_SD(const G4String& name, const G4String &collName);

    //Destructor
    virtual ~NaI_SD();

    // A function that is called at the beginning
    // of each event (OPTIONAL)
    virtual void Initialize(G4HCofThisEvent *hce);

    // A MANDATORY function
    virtual G4bool ProcessHits(G4Step* step, G4TouchableHistory* history);

    // A function that is call at the end of
    // each event (OPTIONAL)
    virtual void EndOfEvent(G4HCofThisEvent *hce);
};
```

Getting information from Sensitive Detector

ProcessHits function will be called for each step.

A Gateway to get the information out from steps

Now we are able to get this information

What all information needs to be stored, depends on the user & application.

All this information can be stored in **HitCollections**

```
#include "NaI_SD.h"
#include "G4Step.hh"
#include "G4Track.hh"
#include "G4SystemOfUnits.hh"

NaI_SD::NaI_SD(const G4String& name, const G4String &collName) :
G4VSensitiveDetector(name) {
collectionName.insert(collName);
}

NaI_SD::~NaI_SD() {}

void NaI_SD::Initialize(G4HCofThisEvent *hce){}

G4bool NaI_SD::ProcessHits(G4Step* step, G4TouchableHistory*) {
    G4Track* track = step->GetTrack();
    G4double energy = track->GetKineticEnergy();
    G4cout << "Detected energy: " << energy / MeV << " MeV" << G4endl;
    return true;
}

void NaI_SD::EndOfEvent(G4HCofThisEvent *hce){}
```

Getting information from Sensitive Detector

ProcessHits function give you information corresponding to current hit.

An Event consist of hundreds of hits (may be more).

The idea is to accumulate information from all hits (in and event) in a sensitive volume.

A Mechanism is required:

(a) User defined mechanism (use c++ array, vector, map, pair etc.) : Possible,

but responsibility of memory allocation and deallocation will be on the user.

If not handled properly may result in memory corruption and abnormal program termination

(b) Geant4 defined Hit collection mechanism : User has to just follow the mechanism, other thing will be taken care by Geant4

Geant4 HitCollection Mechanism

Touching the first templatized class of Geant4

G4THitsCollection : It's basically a container class which can maintain the collection of Geant4 Hits

Eg. std::vector<T> -- std::vector<int>, std::vector<double>, std::vector<std::string>

Now What is Geant4 Hits ???

The definition of Hits is not provided by Geant4 but depends on the user

The set of information you want to get from you sensitive detector constitutes a Hit

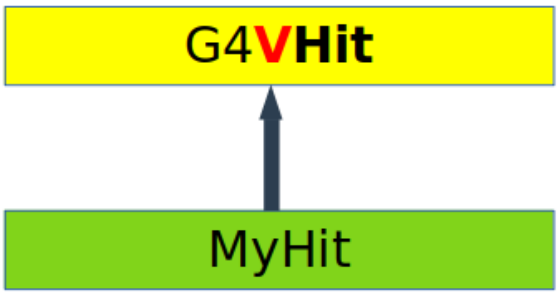
This set of information is encapsulated in C++ class which is known as Hit Class

Hit Class for SensitiveDetector

User has to write a Hit class which must be inherited from G4VHit Class of geant

Here you can store various types of information like:

- **Position**
- **Momentum and energy of track**
- **Geometrical information**
- **Timing information etc..**



Object of Hit class must be stored in the HitCollection

The complete collection is associated to G4Event object via G4HCofThisEvent

These Hit collection are accessible through

- **(a) G4Event object In the EndOfEvent functions**
- **(b) G4SDManager**

Implementation of a Hit Class

A simple implementation of Hit class name **NaI_Hit**

Lets create a **HitCollection** using alias in C++

Better idea would be defined this in separate header file, so that it can be used anywhere.

//HitsCollection.h
using NaiHitCollection = G4THitsCollection<Nai_Hit>

```
#include "G4VHit.hh"

class NaI_Hit : public G4VHit {

public:
  double fStepEnergyDeposited;
  double fStepLength;
  double fStepKineticEnergy;

public:
  NaI_Hit();
  virtual ~NaI_Hit();
  void Fill(double stepEnergy,
            double stepLength,
            double stepKE);
  double GetEnergy() const;
  double GetStepLength() const;
  double GetStepKE() const;
};
```

```
#include "NaI_Hit.h"

NaI_Hit::NaI_Hit() {}

NaI_Hit::~NaI_Hit() {}

void NaI_Hit::Fill(double stepEnergy,
                   double stepLength,
                   double stepKE)
{
  fStepEnergyDeposited = stepEnergy;
  fStepLength          = stepLength;
  fStepKineticEnergy   = stepKE;
}

double NaI_Hit::GetEnergy() const
{
  return fStepEnergyDeposited;
}

double NaI_Hit::GetStepLength() const
{
  return fStepLength;
}

double NaI_Hit::GetStepKE() const
{
  return fStepKineticEnergy;
}
```

Enabling HitCollection in SensitiveDetector

Every HC has 2 parameter: SD name and colleName. This combination is unique for each HC

Geant4 also assigns a unique identified to each collection

```
#include "G4VSensitiveDetector.hh"
#include "G4HCofThisEvent.hh"
#include "HitsCollection.h"

class NaI_SD : public G4VSensitiveDetector {
public:
    NaIHitCollection *fNaIHitCollection;
public:
    NaI_SD(const G4String& name,
           const G4String &collName);

    virtual ~NaI_SD();
    virtual void Initialize(G4HCofThisEvent *hce);
    virtual G4bool ProcessHits(G4Step* step,
                               G4TouchableHistory* history);

    virtual void EndOfEvent(G4HCofThisEvent *hce);
};
```

```
#include "NaI_SD.h"
#include "G4Step.hh"
#include "G4Track.hh"
#include "G4SystemOfUnits.hh"
#include "G4SDManager.hh"
NaI_SD::NaI_SD(const G4String &name, const G4String &collName) : G4VSensitiveDetector
{
  collectionName.insert(collName);
}

NaI_SD::~NaI_SD() {}

void NaI_SD::Initialize(G4HCofThisEvent *hce)
{
  fNaIHitCollection = new NaIHitCollection(SensitiveDetectorName, collectionName[0]);

  G4int hcID        = G4SDManager::GetSDMpointer()->GetCollectionID(collectionName[0]
  hce->AddHitCollection(hcID, fNaIHitCollection);
}

G4bool NaI_SD::ProcessHits(G4Step *step, G4TouchableHistory *)
{
  G4Track *track  = step->GetTrack();
  G4double energy = track->GetKineticEnergy();
  G4cout << "Detected energy: " << energy / MeV << " MeV" << G4endl;
  return true;
}

void NaI_SD::EndOfEvent(G4HCofThisEvent *hce) {}
```

Filling a Hit in the HitCollection

Create an object of Hit class and fill its required data members.

Insert this hit object to the hit collection.

```
#include "NaI_SD.h"
#include "G4Step.hh"
#include "G4Track.hh"
#include "G4SystemOfUnits.hh"
#include "G4SDManager.hh"
#include "Nai_Hit.h"
NaI_SD::NaI_SD(const G4String &name, const G4String &collName) : G4VSensitiveDetector(name)
{
  collectionName.insert(collName);
}

NaI_SD::~NaI_SD() {}

void NaI_SD::Initialize(G4HCofThisEvent *hce)
{
  fNaIHitCollection = new NaIHitCollection(SensitiveDetectorName, collectionName[0]);

  G4int hcID        = G4SDManager::GetSDMpointer()->GetCollectionID(collectionName[0]);
  hce->AddHitCollection(hcID, fNaIHitCollection);
}

G4bool NaI_SD::ProcessHits(G4Step *step, G4TouchableHistory *)
{

  G4Track *track  = step->GetTrack();
  G4double energy = track->GetKineticEnergy();
  G4double stepLength = step->GetStepLength();
  G4double energyDep = step->GetTotalEnergyDeposit();

  Nai_Hit *newHit = new Nai_Hit();
  newHit->Fill(energyDep,stepLength,energy);
  fNaIHitCollection->insert(newHit);
  G4cout << "Detected energy: " << energy / MeV << " MeV" << G4endl;
  return true;
}

void NaI_SD::EndOfEvent(G4HCofThisEvent *hce) {}
```

Processing HitCollection of the complete event

In a simulation we may have multiple hitCollections defined in different sensitive detectors.

Each sensitive detector fill its HitCollection and add it to container of HitsCollection (object of G4HCofThisEvent)

All these hit collections are available in the G4UserEventAction hook, where they can be accessed in the EndOfEventAction function via the object G4Event class.

Use case of multiple hitCollection:

Particularly useful when you want to have **coincidence** between different detector components.

To get the coincidence data, check that the hit collection from these components must have non-zero size, which implies that hit is detected by required detector components.

Processing HitCollection of the complete event

```
#include "G4UserEventAction.hh"
#include "G4Event.hh"
class NaI_EventAction : public G4UserEventAction {
public:
    NaI_EventAction();
    virtual ~NaI_EventAction();
    virtual void BeginOfEventAction(const G4Event *event);
    virtual void EndOfEventAction(const G4Event *event);
};
```

Processing HitCollection of the complete event

Make sure to use the Same collection name that you had assigned in the Initialize function of sensitive detector.

```
#include "NaI_EventAction.h"
#include "G4HCofThisEvent.hh"
#include "G4SDManager.hh"
#include "HitsCollection.h"
NaI_EventAction::NaI_EventAction() {}

NaI_EventAction::~NaI_EventAction() {}

void NaI_EventAction::BeginOfEventAction(const G4Event *event) {}

void NaI_EventAction::EndOfEventAction(const G4Event *event)
{
  G4HCofThisEvent *hce                = event->GetHCofThisEvent();
  G4int hcID                          = G4SDManager::GetSDMpointer()->GetCollectionID("NaiCollection")
;
  NaiHitCollection *naiHitCollection = static_cast<NaiHitCollection *>(hce->GetHC(hcID));

  // Now loop over all the hits in this collection
  for (unsigned int i = 0; i < naiHitCollection->entries(); i++) {
    double energy     = (*naiHitCollection)[i]->GetEnergy();
    double stepLength = (*naiHitCollection)[i]->GetStepLength();
    double ke         = (*naiHitCollection)[i]->GetStepKE();
    //use this data and save it to a file or fill a
    //ROOT tree
  }
}
```

```cpp
#include "G4VSensitiveDetector.hh"
#include "G4HCofThisEvent.hh"

//Public Inheritance
class NaI_SD : public G4VSensitiveDetector {
public:
    //Constructor
    NaI_SD(const G4String& name, const G4String &collName);

    //Destructor
    virtual ~NaI_SD();

    // A function that is called at the beginning
    // of each event (OPTIONAL)
    virtual void Initialize(G4HCofThisEvent *hce);

    // A MANDATORY function
    virtual G4bool ProcessHits(G4Step* step, G4TouchableHistory* history);

    // A function that is call at the end of
    // each event (OPTIONAL)
    virtual void EndOfEvent(G4HCofThisEvent *hce);
};
```