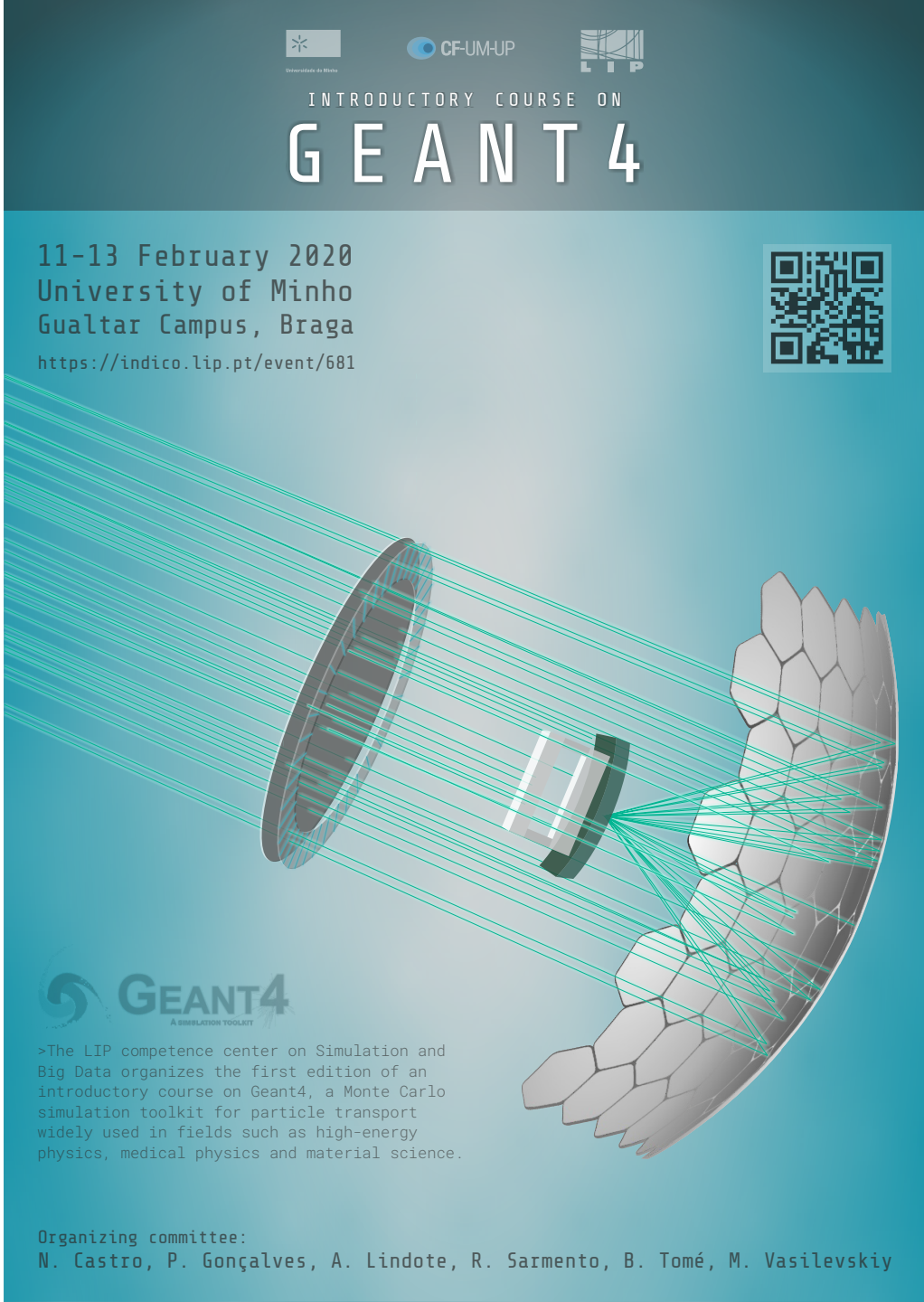
Sensitive Detectors

Patrícia Gonçalves

Slides adapted from slides produced by :
Marc Verderi,Dennis Wright, Vladimir Ivantchenko, Mihaly Novak
http://cern.ch/geant4



Contents

- Sensitive detector and hit
- Hit class
- Sensitive detector class
- G4HCofThisEvent class and its use
- User classes

Sensitive detectors

Extracting useful information

- Given geometry, physics and primary track generation, Geant4 does proper physics simulation **“silently**”.
  - You have to add a bit of code to extract information useful to you.
- There are three ways:
  - **Built-in scoring commands**
    - Most commonly-used physics quantities are available.
  - **Use scorers in the tracking volume**
    - Create scores for each event
    - Create own Run class to accumulate scores
  - **Assign G4VSensitiveDetector to a volume to generate “hits”.**
    - Use user hooks (G4UserEventAction, G4UserRunAction) to get event / run summary
- You may also use user hooks (**G4UserTrackingAction, G4UserSteppingAction**, etc.) :
  - full access to almost all information
  - straight-forward, but do-it-yourself !

Sensitive detector

- A tracker detector typically generates a hit for every single step of every single (charged) track.
  - A tracker hit typically contains
    - Position and time
    - Energy deposition of the step
    - Track ID
- A calorimeter detector typically generates a hit for every cell, and accumulates energy deposition in each cell for all steps of all tracks.
  - A calorimeter hit typically contains
    - Sum of deposited energy
    - Cell ID

Sensitive detector and Hit

- Each Logical Volume can have a pointer to a sensitive detector.
  - Then this volume becomes **sensitive**.
- Hit is a snapshot of the physical interaction of a track or an accumulation of interactions of tracks in the sensitive region of your detector.
- **A sensitive detector creates hit(s)** using the information given in G4Step object. The user has to provide his/her own implementation of the detector response.
  - UserSteppingAction class **should NOT** do this.
- Hit objects, which are still the user's class objects, are collected in a G4Event object at the end of an event.

Hit class

- Hit is a user-defined class derived from **G4VHit**.
- You can store various types of information by implementing your own concrete Hit class.

For example:

  - Position and time of the step
  - Momentum and energy of the track
  - Energy deposition of the step
  - Geometrical information
  - etc, etc, …

Hit class

- Hit is a user-defined class derived from G4VHit.
- You can store various types of information by implementing your own concrete Hit class.

For example:

  - Position and time of the step
  - Momentum and energy of the track
  - Energy deposition of the step
  - Geometrical information
  - etc, etc, …

- Hit objects of a concrete hit class must be stored in a dedicated collection which is instantiated from G4THitsCollection template class.
- The collection will be associated to a G4Event object via G4HCofThisEvent.
- Hits collections are accessible
  - through **G4Event** at the end of event.
    - to be used for analyzing an event
  - through **G4SDManager** during processing an event.
    - to be used for event filtering.

(HCofThisEvent == Hits container of this event)

Hits collection, hits map

- **G4VHitsCollection is the common abstract base class of both G4THitsCollection and G4THitsMap.**
- **G4THitsCollection is a template vector class to store pointers of objects of one concrete hit class type.**
  - **A hit class (derived from G4VHit abstract base class) should have its own identifier (e.g. cell ID).**
  - **In other words, G4THitsCollection requires you to implement your hit class.**
- G4THitsMap is a template map class so that it stores keys (typically cell ID, i.e. copy number of the volume) with pointers of objects of one type.
  - Objects may not be those of hit class.
    - All of currently provided scorer classes use G4THitsMap with simple double.
  - Since G4THitsMap is a template, it can be used by your sensitive detector class to store hits.

Implementation of Hit class

```
#include "G4VHit.hh"
class Hit : public G4VHit
{
  public:
      Hit(some_arguments);
      virtual ~Hit();
      virtual void Draw();
      virtual void Print();
  private:
      // some data members
  public:
      // some set/get methods
};

#include “G4THitsCollection.hh”
typedef G4THitsCollection<Hit> SDHitCollection;
```

Sensitive Detector class

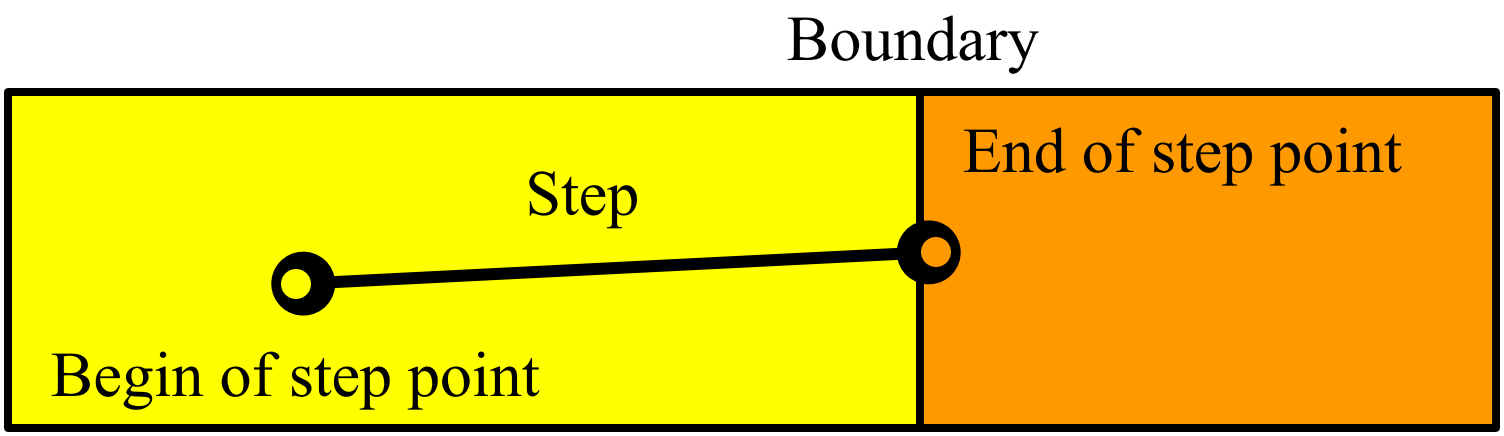
- **Sensitive detector is a user-defined class derived from G4VSensitiveDetector**

```
#include "G4VSensitiveDetector.hh"
#include "Hit.hh”

class SensitiveDetector : public G4VSensitiveDetector
{
  public:
    SensitiveDetector(G4String SDname);
    ~SensitiveDetector();

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

  private:
  SDHitCollection* fHitCollection;
};
```

Detector sensitivity

- A sensitive detector either
  - constructs one or more hit objects or
  - accumulates values to existing hits

  using information given in a **G4Step** object.
- Note that you must get the **volume information** from the “PreStepPoint”.



Implementation of Sensitive Detector

SensitiveDetector.cc

```
#include "SensitiveDetector.hh"

SensitiveDetector::SensitiveDetector(G4String SDname)
: G4VSensitiveDetector(SDname)
{
  G4cout << "Creating SD with name: " << SDname<<G4endl;

  collectionName.insert("SDHitCollection");
 }
```

- In the constructor, define the name of the hits collection which is handled by this sensitive detector

- In case your sensitive detector generates more than one kinds of hits (e.g. anode and cathode hits separately), define all collection names.

Implementation of Sensitive Detector

SensitiveDetector.cc

```
void SensitiveDetector::Initialize(G4HCofThisEvent*HCE)
{
  fHitCollection = new SDHitCollection (GetName(), collectionName[0]);
  static G4int HCID = -1;
  if (HCID<0) {
    HCID = GetCollectionID(0);
  }
  HCE->AddHitsCollection(HCID, fHitCollection);
}
```

- Initialize() method is invoked **at the beginning of each event.**
- Get the unique ID number for this collection.
  - **GetCollectionID()** is a heavy operation. It should not be used for every event.
  - **GetCollectionID()** is available after this sensitive detector object is registered to G4SDManager. **Thus, this method cannot be used in the constructor of this detector class.**
- Instantiate hits collection(s) and attach to **G4HCofThisEvent** object given in the argument.

Implementation of Sensitive Detector

SensitiveDetector.cc

```
G4bool SensitiveDetector::ProcessHits(G4Step *aStep, G4TouchableHistory *)
{
  … see the code

  class Hit * hit = new class Hit();

   …

  return true;
}
```

- The **ProcessHits()** method is invoked **for every step** in the volume(s) where this sensitive detector is assigned.
- In this method, generate a hit corresponding to the current step (for tracking detector), or accumulate the energy deposition of the current step to the existing hit object where the curren step belongs to (for calorimeter detector).
- **Don't forget to collect geometry information (e.g. copy number) from PreStepPoint.**

Implementation of Sensitive Detector

SensitiveDetector.cc

```
void SensitiveDetector::EndOfEvent(G4HCofThisEvent*)
{
//    fHitCollection->PrintAllHits();
}
```

- This method is invoked at the end of processing an event.
  - It is invoked even if the event is aborted.
  - It is invoked before **UserEndOfEventAction**.

Step point and Touchable

- **As mentioned already, G4Step has two G4StepPoint objects as its starting and ending points. All the geometrical information of the particular step should be taken from "PreStepPoint".**
  - **Geometrical information associated with G4Track is identical to "PostStepPoint".**
- **Each G4StepPoint object has**
  - **Position in world coordinate system**
  - **Global and local time**
  - **Material**
  - **G4TouchableHistory for geometrical information**
- **G4TouchableHistory object is a vector of information for each geometrical hierarchy.**
  - **copy number**
  - **transformation / rotation to its mother**

Touchable

- **G4TouchableHistory has information of geometrical hierarchy of the point.**

```
G4Step* aStep;
G4StepPoint* preStepPoint = aStep->GetPreStepPoint();
G4TouchableHistory* theTouchable =
    (G4TouchableHistory*)(preStepPoint->GetTouchable());
G4int copyNo = theTouchable->GetVolume()->GetCopyNo();
G4int motherCopyNo
            = theTouchable->GetVolume(1)->GetCopyNo();
G4int grandMotherCopyNo
            = theTouchable->GetVolume(2)->GetCopyNo();
G4ThreeVector worldPos = preStepPoint->GetPosition();
G4ThreeVector localPos = theTouchable->GetHistory()
    ->GetTopTransform().TransformPoint(worldPos);
```

Defining a sensitive detector

DetectorConstruction.cc

```
// Create a sensitive detector
  G4SDManager* SDmanager = G4SDManager::GetSDMpointer();

  SensitiveDetector* SD = new SensitiveDetector ("SD1");
  SDmanager->AddNewDetector(SD);
  myDetector_log->SetSensitiveDetector(SD);
```

- Each sensitive detector (SD) object must have a unique name.
  - Some logical volumes can share one detector object (SD).
  - More than one detector objects (SD) can be made from one detector class with different detector name (SD).
  - One logical volume cannot have more than one SD detector objects.
  - But, as discussed before, one SD can generate more than one kinds of hits.
    - e.g. a drift chamber class may generate anode and cathode hits separately.

G4HCofThisEvent

- A G4Event object has a **G4HCofThisEvent** object at the end of (successful) event processing. G4HCofThisEvent object stores all hits collections made within the event.
  - Pointer(s) to the collections may be NULL if collections are not created in the particular event.
  - Hits collections are stored by pointers of G4VHitsCollection base class. Thus, you have to **cast** them to the types of your individual concrete classes.
  - The index number of a Hits collection is unique and unchanged for a run. The index number can be obtained by

    `G4SDManager::GetCollectionID("`*`detName/colName`***`");`**

    - The index table is also stored in G4Run.

Usage of G4HCofThisEvent

EventAction.cc

```
void EventAction::EndOfEventAction(const G4Event* event)
{
… see code

  G4HCofThisEvent* HCE = event->GetHCofThisEvent();
  SDHitCollection * Hits = 0;
  Hits = GetHitCollection(HCE,"SDHitCollection"); // Particles detected at detector

  if (Hits) {
    const G4int nHits = Hits->entries();
    for (G4int iHit = 0; iHit<nHits; ++iHit) {
      outFile << (*Hits)[iHit] << G4endl;
    }
  }
  else
    G4Exception("EndOfEventAction","",JustWarning,"Hits collection SDHitCollection not
    found.");
}

SDHitCollection * EventAction::GetHitCollection(G4HCofThisEvent* HCE, const G4String & name)
{
  static G4SDManager * SDman = G4SDManager::GetSDMpointer();
  const G4int HitCollID  = SDman->GetCollectionID(name);
  if (HCE && HitCollID > -1)
    return (SDHitCollection*)(HCE->GetHC(HitCollID));
  else
    return 0;
}
```

cast !