Read-out and detector response

http://cern.ch/geant4

The full set of lecture notes of this Geant4 Course is available at http://www.ge.infn.it/geant4/events/nss2003/geant4course.html
Contents

- Sensitive detector and hit
- Digitizer module and digit
- Hit class
- Sensitive detector class
- Touchable
- Readout geometry
- G4HCofThisEvent class and its use
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.
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”.

![Diagram showing the steps of a Geant4 simulation](image-url)
Digitizer module and digit

- Digit represents a detector output (e.g. ADC/TDC count, trigger signal, etc.).

- Digit is created with one or more hits and/or other digits by a user's concrete implementation derived from G4VDigitizerModule.

- In contradiction to the sensitive detector which is accessed at tracking time automatically, the digitize() method of each G4VDigitizerModule must be explicitly invoked by the user’s code (e.g. at EventAction).
**Hit class**

- Hit is a user-defined class derived from G4VHit.
- You can store various types 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
  - or any combination of above

- 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 G4HCoofThisEvent.
- 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.
Implementation of Hit class

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

#include "G4THitsCollection.hh"
typedef G4THitsCollection<MyDriftChamberHit> MyDriftChamberHitsCollection;
```
Sensitive Detector class

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

```cpp
#include "G4VSensitiveDetector.hh"
#include "MyDriftChamberHit.hh"
class G4Step;
class G4HCofThisEvent;
class MyDriftChamber : public G4VSensitiveDetector
{
    public:
        MyDriftChamber(G4String name);
        virtual ~MyDriftChamber();
        virtual void Initialize(G4HCofThisEvent*HCE);
        virtual G4bool ProcessHits(G4Step*aStep,
                                  G4TouchableHistory*ROhist);
        virtual void EndOfEvent(G4HCofThisEvent*HCE);
    
    private:
        MyDriftChamberHitsCollection * hitsCollection;
        G4int collectionID;
};
```
Implementation of Sensitive Detector

MyDriftChamber::MyDriftChamber(G4String name)
    :G4VSensitiveDetector(name)
{ collectionName.insert("driftChamberCollection");
    collectionID = -1; }

void MyDriftChamber::Initialize(G4HCofThisEvent*HCE)
{ hitsCollection = new MyDriftChamberHitsCollection
    (SensitiveDetectorName, collectionName[0]);
    if(collectionID<0)
    { collectionID = G4SDManager::GetSDMpointer() - >GetCollectionID(hitsCollection); }
    HCE->AddHitsCollection(collectionID,hitsCollection); }

G4bool MyDriftChamber::ProcessHits
   (G4Step*aStep,G4TouchableHistory*ROhist)
{ MyDriftChamberHit* aHit = new MyDriftChamberHit();
   // some set methods
   ...
   hitsCollection->insert(aHit);
   return true; }

void MyDriftChamber::EndOfEvent(G4HCofThisEvent*HCE) {;}

Geant4 Training 2003
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 basically same as “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
Copy number

- Suppose a calorimeter is made of 4x5 cells.
  - and it is implemented by two levels of replica.

- In reality, there is only one physical volume object for each level. Its position is parameterized by its copy number.

- To get the copy number of each level, suppose what happens if a step belongs to two cells.
  - Remember geometrical information in G4Track is identical to "PostStepPoint". You cannot get the collect copy number for "PreStepPoint" if you directly access to the physical volume.

- Use touchable to get the proper copy number, transform matrix, etc.
Touchable

- G4TouchableHistory has information of geometrical hierarchy of the point.

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

- In some cases of most complicated geometries, it is not easy to define volume boundaries corresponding to the readout segmentation.
- Readout geometry is a **virtual** and **artificial** geometry which can be defined **in parallel to the real** detector geometry.
- Readout geometry is optional. May have more than one.
  - Each one should be associated to a sensitive detector.
- Note that a step is **not** limited by the boundary of readout geometry.
Defining a sensitive detector

- Basic strategy
  
  ```
  G4LogicalVolume* myLogCalor = ....;
  G4VSensitiveDetector* pSensetivePart = 
      new MyCalorimeter("/mydet/calorimeter1");
  G4SDManager* SDMan = G4SDManager::GetSDMpointer();
  SDMan->AddNewDetector(pSensetivePart);
  myLogCalor->SetSensitiveDetector(pSensetivePart);
  ```

- Each detector **object** must have a unique name.
  - Some logical volumes can share one detector object
  - More than one detector objects can be made from one detector class with different detector name.
  - One logical volume cannot have more than one detector objects. But, one detector object 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) may be NULL if collection(s) are not created in the particular event.
  - Hits collections are stored by pointers of G4VHitsCollection base class. Thus, you have to cast them to types of individual concrete classes.
Usage of G4HCofThisEvent

```cpp
int CHCID = G4SDManager::GetSDMpointer()
    ->GetCollectionID("myDet/calorimeter1/collection1");
G4HCofThisEvent* HCE = evt->GetHCofThisEvent();
MyCalorimeterHitsCollection* CHC = 0;
if(HCE)
{CHC = (MyCalorimeterHitsCollection*)(HCE->GetHC(CHCID));}
if(CHC)
{ int n_hit = CHC->entries();
  G4cout<<"Calorimeter has "<<n_hit<<" hits."<<G4endl;
  for(int i1=0;i1<n_hit;i1++)
  { MyCalorimeterHit* aHit = (*CHC)[i1];
    aHit->Print(); } }
```

- This scheme can be utilized also for Digitization.