Scoring II

Makoto Asai (SLAC)

Geant4 Tutorial Course

Geant4

Contents

- Sensitive detector and hit
- Hit class
- Sensitive detector class
- Touchable
- Use of G4HCofThisEvent class

Sensitive detector and hit

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.
- Hit objects, which are still the user's class objects, are collected in a G4Event object at the end of an event.

Sensitive detector vs. primitive scorer

Sensitive detector

- You have to implement your own detector and hit classes.
- One hit class can contain many quantities. A hit can be made for each individual step, or accumulate quantities.
- Basically one hits collection is made per one detector.
- Hits collection is relatively compact.

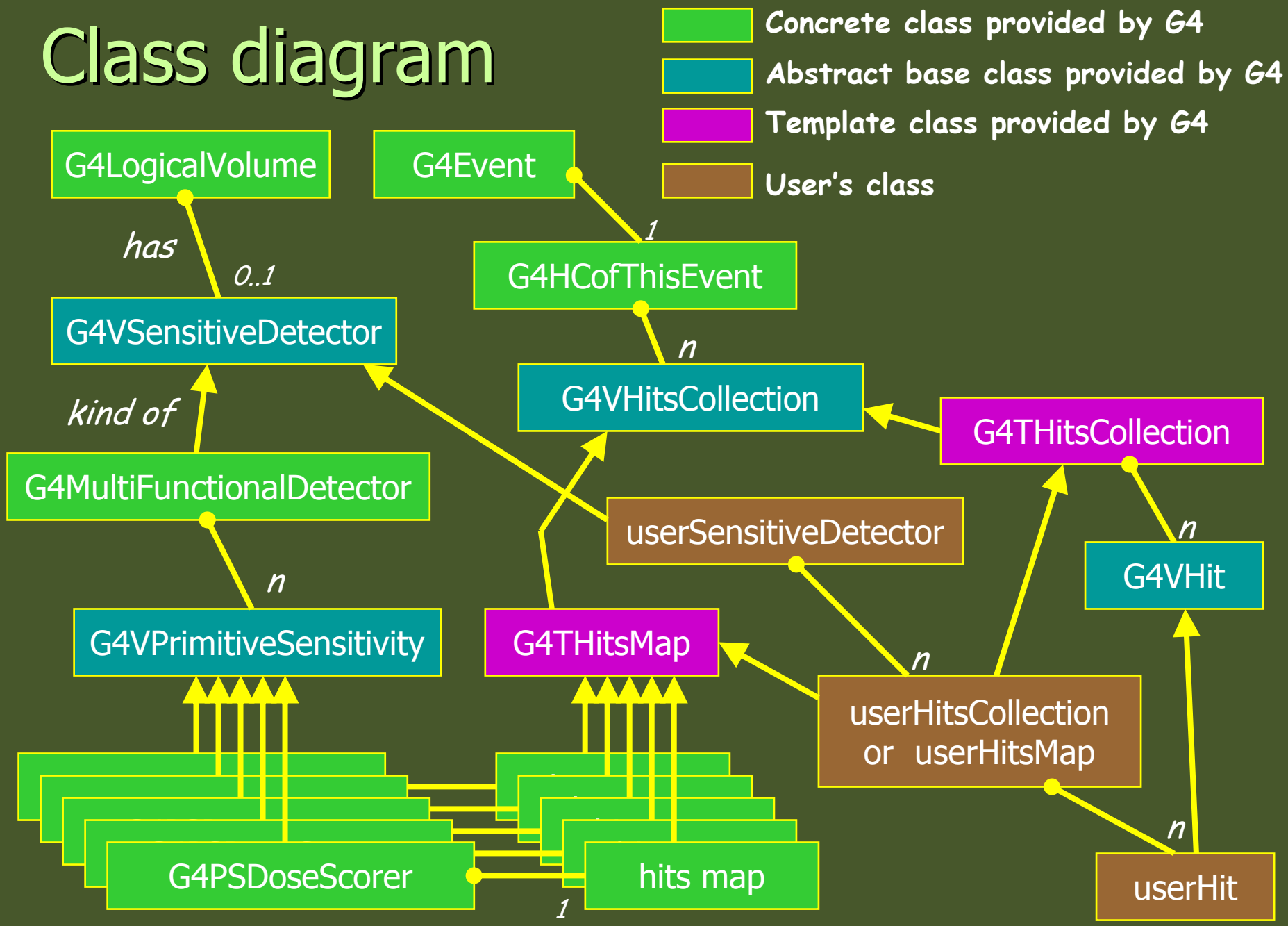
Primitive scorer

- Many scorers are provided by Geant4. You can add your own.
- Each scorer accumulates one quantity for an event.
- G4MultiFunctionalDetector creates many collections (maps), i.e. one collection per one scorer.
- Keys of maps are redundant for scorers of same volume.

I would suggest to :

- Use primitive scorers
  - if you are not interested in recording each individual step but accumulating some physics quantities for an event for a run, and
  - if you do not have to have too many scorers.
- Otherwise, consider implementing your own sensitive detector.

Class diagram



Hit class

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 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.

Implementation of Hit class

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

#include “G4THitsCollection.hh”
typedef G4THitsCollection<MyHit> MyHitsCollection;
```

Sensitive detector class

Sensitive Detector class

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

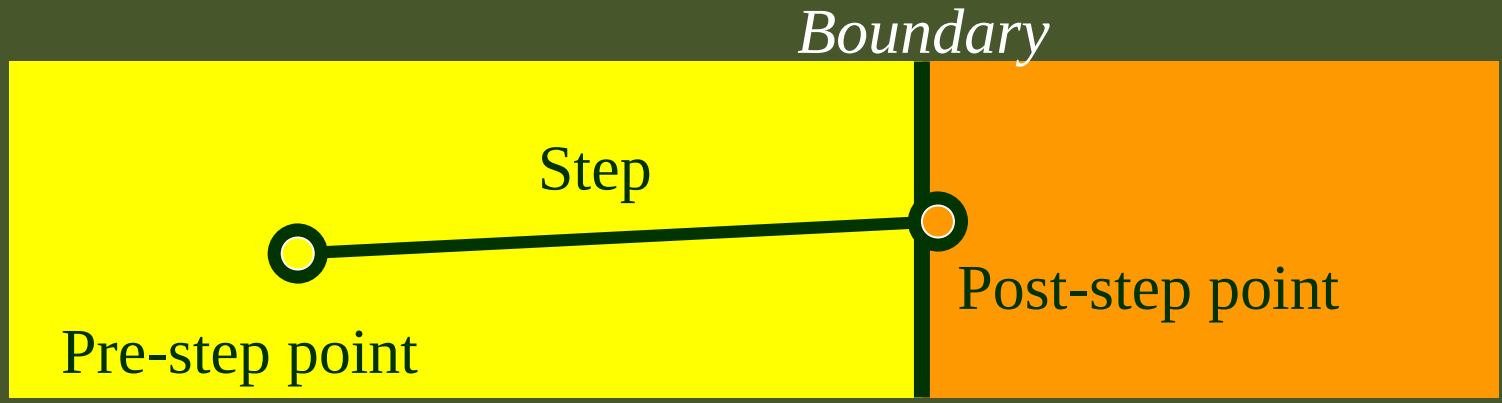
```
#include "G4VSensitiveDetector.hh"
#include "MyHit.hh"
class G4Step;
class G4HCofThisEvent;
class MyDetector : public G4VSensitiveDetector
{
  public:
      MyDetector(G4String name);
      virtual ~MyDetector();
      virtual void Initialize(G4HCofThisEvent*HCE);
      virtual G4bool ProcessHits(G4Step*aStep,
                                 G4TouchableHistory*ROhist);
      virtual void EndOfEvent(G4HCofThisEvent*HCE);
  private:
      MyHitsCollection * hitsCollection;
      G4int collectionID;
};
```

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
- You can instantiate more than one objects for one sensitive detector class. Each object should have its unique detector name.
  - For example, each of two sets of detectors can have their dedicated sensitive detector objects. But, the functionalities of them are exactly the same to each other so that they can share the same class. See examples/extended/analysis/A01 as an example.

Step

- Step has two points and also "delta" information of a particle (energy loss on the step, time-of-flight spent by the step, etc.).
- Each point knows the volume (and material). In case a step is limited by a volume boundary, the end point physically stands on the boundary, and it logically belongs to the next volume.
- Note that you must get the volume information from the "PreStepPoint".



Implementation of Sensitive Detector - 1

```
MyDetector::MyDetector(G4String detector_name)
                  :G4VSensitiveDetector(detector_name),
                   collectionID(-1)
{
  collectionName.insert("collection_name");
}
```

- 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 - 2

```
void MyDetector::Initialize(G4HCofThisEvent*HCE)
{
  if(collectionID<0) collectionID = GetCollectionID(0);
  hitsCollection = new MyHitsCollection
        (SensitiveDetectorName,collectionName[0]);
  HCE->AddHitsCollection(collectionID,hitsCollection);
}
```

- 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 events.
  - GetCollectionID() is available after this sensitive detector object is constructed and registered to G4SDManager. Thus, this method cannot be invoked in the constructor of this detector class.
- Instantiate hits collection(s) and attach it/them to G4HCofThisEvent object given in the argument.
- In case of calorimeter-type detector, you may also want to instantiate hits for all calorimeter cells with zero energy depositions, and insert them to the collection.

Implementation of Sensitive Detector - 3

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

- This ProcessHits() method is invoked for every steps 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 current step belongs to (for calorimeter detector).
- Don't forget to collect geometry information (e.g. copy number) from "PreStepPoint".
- Currently, returning boolean value is not used.

Implementation of Sensitive Detector - 4

```
void MyDetector::EndOfEvent(G4HCofThisEvent*HCE)
{;}
```

- 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.

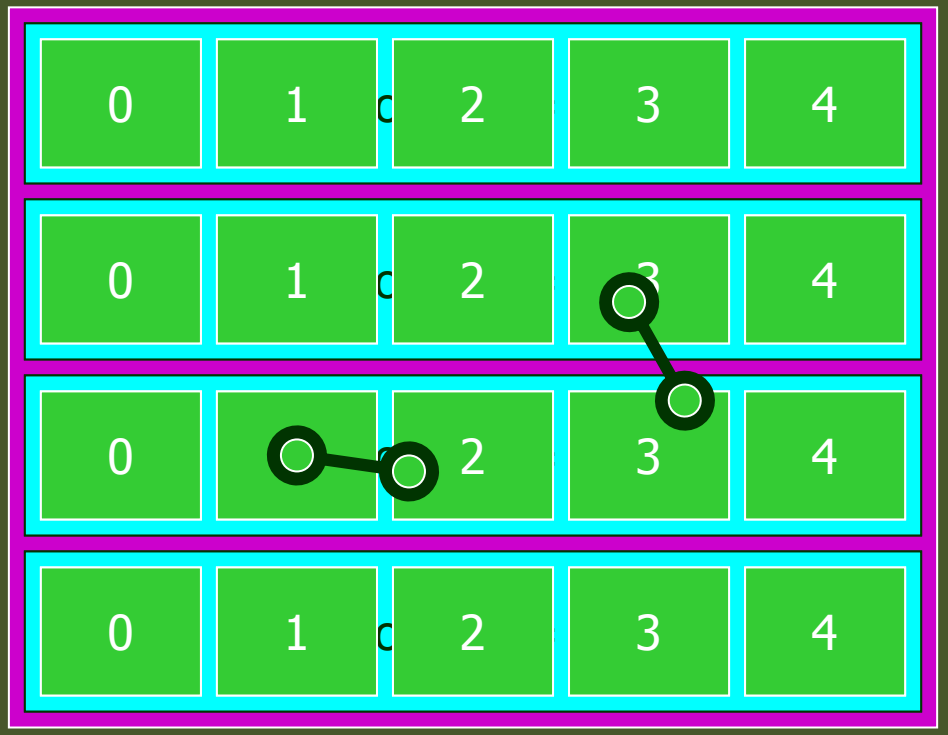
Touchable

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
- Since release 4.0, *handles* (or *smart-pointers*) to touchables are intrinsically used. Touchables are reference counted.

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 correct 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.

```
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);
```

Use of G4HCofThisEvent class

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 types of 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

```
void MyEventAction::EndOfEventAction(const G4Event* evt)
{
 static int CHCID = -1;
 If(CHCID<0) CHCID = G4SDManager::GetSDMpointer()
   ->GetCollectionID("myDet/collection1");
 G4HCofThisEvent* HCE = evt->GetHCofThisEvent();
 MyHitsCollection* CHC = 0;
 if(HCE)
 { CHC = (MyHitsCollection*)(HCE->GetHC(CHCID)); }
 if(CHC)
 {
  int n_hit = CHC->entries();
  G4cout<<“My detector has ”<<n_hit<<" hits."<<G4endl;
  for(int i1=0;i1<n_hit;i1++)
  {
    MyHit* aHit = (*CHC)[i1];
    aHit->Print();
  }
 }
}
```

Cast !

When to invoke GetCollectionID()?

- Which is the better place to invoke G4SDManager::GetCollectionID() in a user event action class, in its constructor or in the BeginOfEventAction()?
- It actually depends on the user's application.
  - Note that construction of sensitive detectors (and thus registration of their hits collections to SDManager) takes place when the user issues RunManager::Initialize(), and thus the user's geometry is constructed.
- In case user's EventAction class should be instantiated before Runmanager::Initialize() (or /run/initialize command), GetCollectionID() should not be in the constructor of EventAction.
- While, if the user has nothing to do to Geant4 before RunManager::Initialize(), this initialize method can be hard-coded in the main() before the instantiation of EventAction (e.g. exampleA01), so that GetCollectionID() could be in the constructor.