Hits

Geant4 CERN Tutorial

Wednesday, 17 February 2010

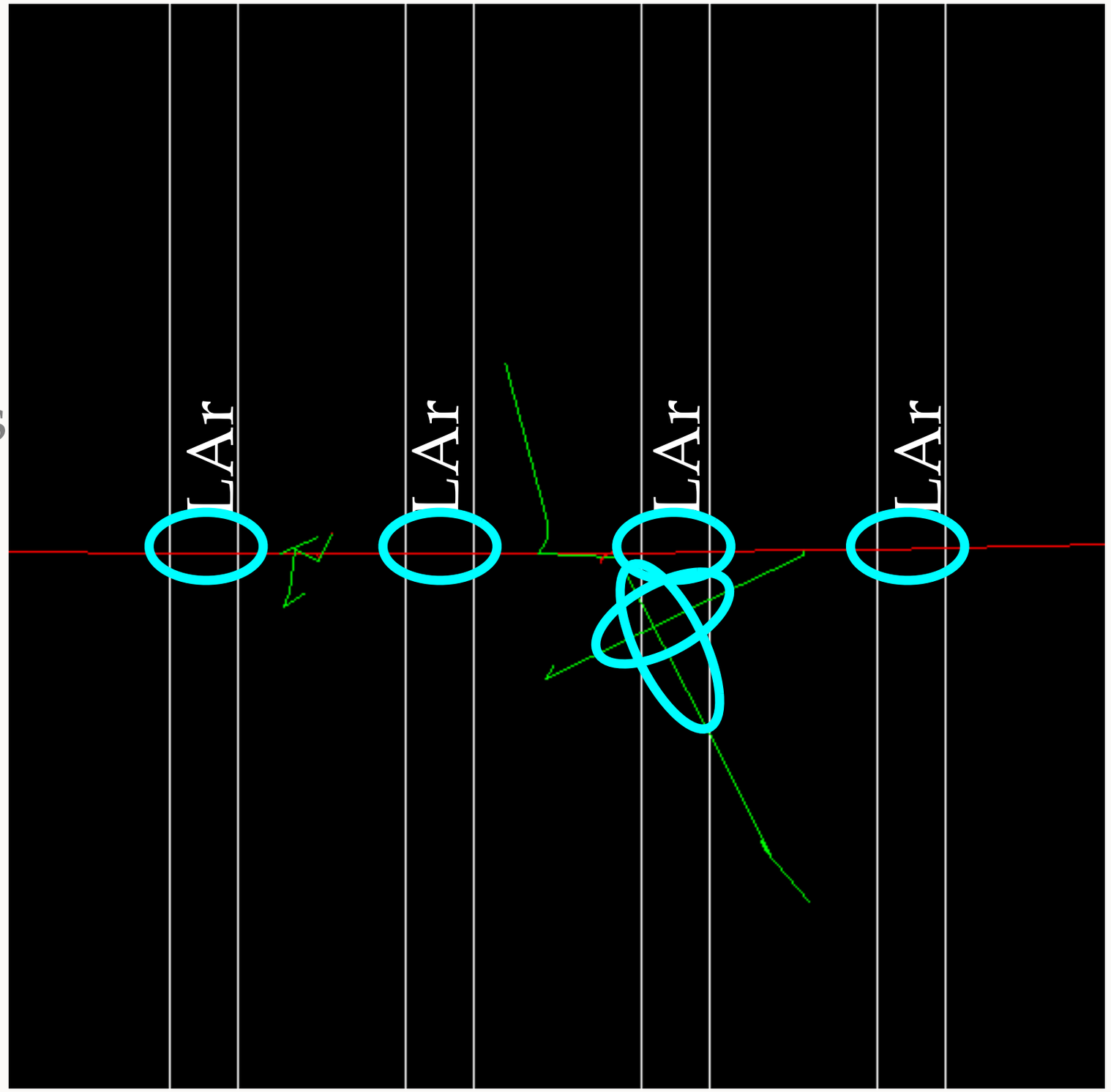
In This Lecture

- What is G4Hit?
- How to use hits

Important: this material refers to this morning lecture: Sensitive Detector

Reminder: G4Step

For all these G4Steps ::ProcessHits(...) will be called



What Hits Are

- Hits are created in Sensitive Detector to store user quantities
- Hits are collected in a container and "registered" in Geant4
  - Hits become available to all components of the application
- 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
- Hits should be identified: they have an id that uniquely identifies them

Hits

- You need to write your own Hit class: inherits from G4VHit
- Hits must be stored in a collection of hits instantiated from G4THitsCollection template class

```
#include "G4VHit.hh"
#include "G4Allocator.hh"
#include "G4THitsCollection.hh"
```

← Headers files

```
class HadCaloHit : public G4VHit {
public:
  HadCaloHit(const G4int layer);
 ~HadCaloHit();
  void Print();
  void          AddEdep(const double e){ eDep += e; }

  G4double      GetEdep()        const { return eDep;}
  G4int         GetLayerNumber() const { return layerNumber; }
private:
  const G4int   layerNumber;
  G4double      eDep;
};

// Define the "hit collection" using the template class G4THitsCollection:
typedef G4THitsCollection<HadCaloHit> HadCaloHitCollection;
```

Hits

- You need to write your own Hit class: inherits from G4VHit
- Hits must be stored in a collection of hits instantiated from G4THitsCollection template class

```
#include "G4VHit.hh"
#include "G4Allocator.hh"
#include "G4THitsCollection.hh"

class HadCaloHit : public G4VHit {
public:
  HadCaloHit(const G4int layer);
 ~HadCaloHit();
  void Print();
  void          AddEdep(const double e){ eDep += e; }

  G4double      GetEdep()        const { return eDep;}
  G4int         GetLayerNumber() const { return layerNumber; }
private:
  const G4int   layerNumber;
  G4double      eDep;
};

// Define the "hit collection" using the template class G4THitsCollection:
typedef G4THitsCollection<HadCaloHit> HadCaloHitCollection;
```

Base class

Hits

- You need to write your own Hit class: inherits from G4VHit
- Hits must be stored in a collection of hits instantiated from G4THitsCollection template class

```
#include "G4VHit.hh"
#include "G4Allocator.hh"
#include "G4THitsCollection.hh"

class HadCaloHit : public G4VHit {
public:
  HadCaloHit(const G4int layer);
  ~HadCaloHit();
  void Print();
  void           AddEdep(const double e){ eDep += e; }

  G4double       GetEdep()        const { return eDep;}
  G4int          GetLayerNumber() const { return layerNumber; }
private:
  const G4int   layerNumber;
  G4double      eDep;
};

// Define the "hit collection" using the template class G4THitsCollection:
typedef G4THitsCollection<HadCaloHit> HadCaloHitCollection;
```

Create a new Hit: the ID is the layer index

Hits

- You need to write your own Hit class: inherits from G4VHit
- Hits must be stored in a collection of hits instantiated from G4THitsCollection template class

```
#include "G4VHit.hh"
#include "G4Allocator.hh"
#include "G4THitsCollection.hh"

class HadCaloHit : public G4VHit {
public:
  HadCaloHit(const G4int layer);
  ~HadCaloHit();
  void Print();
  void                AddEdep(const double e){ eDep += e; }

  G4double      GetEdep()        const { return eDep;}
  G4int         GetLayerNumber() const { return layerNumber; }
private:
  const G4int   layerNumber;
  G4double      eDep;
};

// Define the "hit collection" using the template class G4THitsCollection:
typedef G4THitsCollection<HadCaloHit> HadCaloHitCollection;
```

Hit interface: print on screen

Hits

- You need to write your own Hit class: inherits from G4VHit
- Hits must be stored in a collection of hits instantiated from G4THitsCollection template class

```
#include "G4VHit.hh"
#include "G4Allocator.hh"
#include "G4THitsCollection.hh"

class HadCaloHit : public G4VHit {
public:
  HadCaloHit(const G4int layer);
 ~HadCaloHit();
  void Print();
  void           AddEdep(const double e){ eDep += e; }

  G4double       GetEdep()        const { return eDep;}
  G4int          GetLayerNumber() const { return layerNumber; }
private:
  const G4int   layerNumber;
  G4double      eDep;
};

// Define the "hit collection" using the template class G4THitsCollection:
typedef G4THitsCollection<HadCaloHit> HadCaloHitCollection;
```

Our specific hit interface

Hits

- You need to write your own Hit class: inherits from G4VHit
- Hits must be stored in a collection of hits instantiated from G4THitsCollection template class

```
#include "G4VHit.hh"
#include "G4Allocator.hh"
#include "G4THitsCollection.hh"

class HadCaloHit : public G4VHit {
public:
  HadCaloHit(const G4int layer);
 ~HadCaloHit();
  void Print();
  void          AddEdep(const double e){ eDep += e; }

  G4double      GetEdep()        const { return eDep;}
  G4int         GetLayerNumber() const { return layerNumber; }
private:
  const G4int   layerNumber;
  G4double      eDep;
};

// Define the "hit collection" using the template class G4THitsCollection:
typedef G4THitsCollection<HadCaloHit> HadCaloHitCollection;
```

The Hit container, just add this line

Warning: more advanced code (memory management optimization) not shown here, optional but highly recommended

How To Declare Hits

- A hits collection has a name, this name must be declared in SensitiveDetector constructor
- SD has a data member: collectionName, add your name to this vector of names

  - A SD can declare more than one hits collection!

```
HadCaloSensitiveDetector::HadCaloSensitiveDetector(G4String SDname)
  : G4VSensitiveDetector(SDname)
{
    G4cout<<"Creating SD with name: "<<SDname<<G4endl;
  // 'collectionName' is a protected data member of base class G4VSensitiveDetector.
  // Here we declare the name of the collection we will be using.
  collectionName.insert("HadCaloHitCollection");

  // Note that we may add as many collection names we wou  ish: ie
  // a sensitive detector can have many collections.
}
```

Our hits collection name!

How To Create A HC (Hits Container)

- Every event a new hit collection (HC) has to be created and added to current event collection of hits
- Every HC has two names: the SD name that created it and the name of collection. This pair is unique
  - Geant4 uses also an identifier (a number) to uniquely identify your collection, you need to use this ID to register/retrieve the collection

```
void HadCaloSensitiveDetector::Initialize(G4HCofThisEvent* HCE)
{
   hitCollection = new HadCaloHitCollection(GetName(), collectionName[0]);
   static G4int HCID = -1;
   if (HCID<0) HCID = GetCollectionID(0); // <<-- this is to get an ID for collectionName[0]
   HCE->AddHitsCollection(HCID, hitCollection);
```

How To Create A HC (Hits Container)

- Every event a new hit collection (HC) has to be created and added to current event collection of hits

- Every HC has two names: the SD name that created it and the name of collection. This pair is unique

  - Geant4 uses also an identifier (a number) to uniquely identify your collection, you need to use this ID to register/retrieve the collection

```
void HadCaloSensitiveDetector::Initialize(G4HCofThisEvent* HCE)
{
  hitCollection = new HadCaloHitCollection(GetName(), collectionName[0]);
  static G4int HCID = -1;
  if (HCID<0) HCID = GetCollectionID(0); // <<-- this is to get an ID for collectionName[0]
  HCE->AddHitsCollection(HCID, hitCollection);
```

Create a Hit Collection: GetName() returns SD name (“/HadCalo”) , collectionName is a vector: [0] is the first (and only in our case) element (“HadCaloHitCollection”)

How To Create A HC (Hits Container)

- Every event a new hit collection (HC) has to be created and added to current event collection of hits

- Every HC has two names: the SD name that created it and the name of collection. This pair is unique

  - Geant4 uses also an identifier (a number) to uniquely identify your collection, you need to use this ID to register/retrieve the collection

```
void HadCaloSensitiveDetector::Initialize(G4HCofThisEvent* HCE)
{
   hitCollection = new HadCaloHitCollection(SDName(), collectionName[0]);
   static G4int HCID = -1;
   if (HCID<0) HCID = GetCollectionID(0); // <<-- this is to get an ID for collectionName[0]
   HCE->AddHitsCollection(HCID, hitCollection);
}
```

GetCollectionID(0) is an heavy operation, you should avoid to do it every event!
GetCollectionID(0) returns the unique ID associated to the hit collection!

How To Create A HC (Hits Container)

- Every event a new hit collection (HC) has to be created and added to current event collection of hits

- Every HC has two names: the SD name that created it and the name of collection. This pair is unique

  - Geant4 uses also an identifier (a number) to uniquely identify your collection, you need to use this ID to register/retrieve the collection

```
void HadCaloSensitiveDetector::Initialize(G4HCofThisEvent* HCE)
{
   hitCollection = new HadCaloHitCollection(GetName(), collectionName[0]);
   static G4int HCID = -1;
   if (HCID<0) HCID = GetCollectionID(0); // <<-- this is to get an ID for collectionName[0]
   HCE->AddHitsCollection(HCID, hitCollection);
```

Register the hits collection object in the Hits Collections of This Event (G4HCofThisEvent)

How To Create And Fill Hits

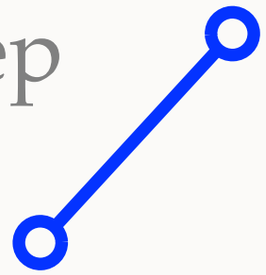
- Every time ProcessHits is called you can (if needed) create a hit and add it to the hits collection

```
G4bool HadCaloSensitiveDetector::ProcessHits(G4Step *step, G4TouchableHistory *)
{
      HadCaloHit* aHit = new HadCaloHit(layerIndex);
     hitCollection->insert(aHit);
     aHit->AddEdep( edep );
     return true;
}
```

- Note: our calorimetric hits accumulate energy on layers, thus hits must be created only the first time a G4Step is in layer "n". You need a way to "remember" if hit has been already created (not shown here)
  - A possible solution is shown in the code of exercise. Not discussed here (use C++ std::map). You are free to use what you want, but try to optimize the code! These part of the code will be called several times during execution.

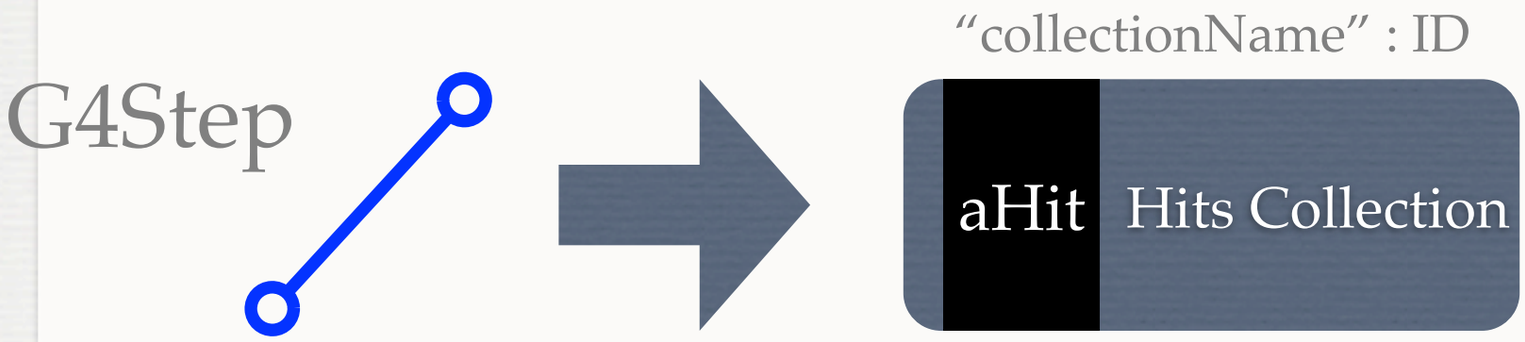
Summary

G4Step



Summary

Summary



Repeat for each step in the event

Summary

End of the event

Exercise

- task4c
  - Modify current Hit class (add print on screen)
  - Create a Hits collection and register it
  - (optional) Extend Hit class
- cd g4course2010/task4/task4c

A Note

- The code of task2a contains example of a “tracker type” hits:
  - Creates a hit for each track passes in a Si module
- It also contain the “digitization” (not discussed here): the process of simulating the electronic read-out of the detector
  - SiDigitizer class: gaussian noise, cross-talk