Makoto Asai (SLAC)
Geant4 Users Workshop @ SLAC
Feb. 18th, 2002
Detector sensitivity

- A logical volume becomes sensitive if it has a pointer to a concrete class derived from G4VSensitiveDetector.

- A sensitive detector either
  - constructs one or more hit objects or
  - accumulates values to existing hits using information given in a G4Step object.

NOTE: you must get the volume information from the “PreStepPoint”.
Sensitive detector and Hit

- Each “Logical Volume” can have a pointer to a sensitive detector.
- 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 still are the user’s class objects, are collected in a G4Event object at the end of an event.
Digitization

- Digit represents a detector output (e.g. ADC/TDC count, trigger signal).
- Digit is created with one or more hits and/or other digits by a concrete implementation derived from G4VDigitizerModule.
- In contradiction to the Hit which is generated 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 class

- 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,
  - through G4SDManager during processing an event.

--> Used for Event filtering.
# Implementation of Hit class

```cpp
#include "G4VHit.hh"
#include "G4ThreeVector.hh"

class T01DriftChamberHit : public G4VHit
{
    public:
        T01DriftChamberHit();
        T01DriftChamberHit(G4int z);
        virtual ~T01DriftChamberHit();
        T01DriftChamberHit(const T01DriftChamberHit &right);
        const T01DriftChamberHit&
            operator=(const T01DriftChamberHit &right);
        int operator==(const T01DriftChamberHit &right) const;
        inline void *operator new(size_t);
        inline void operator delete(void *aHit);
        virtual void Draw();
        virtual void Print();
};
```
Implementation of Hit class

private:
    G4int layerID;
    G4double time;
    G4ThreeVector localPos;
    G4ThreeVector worldPos;

public:
    // some set/get methods

};

#include "G4THitsCollection.hh"

typedef G4THitsCollection<T01DriftChamberHit> T01DriftChamberHitsCollection;
Implementation of Hit class

#include "G4Allocator.hh"

extern G4Allocator<T01DriftChamberHit>
    T01DriftChamberHitAllocator;

inline void* T01DriftChamberHit::operator new(size_t)
{
    void* aHit;
    aHit = (void*)T01DriftChamberHitAllocator.MallocSingle();
    return aHit;
}

inline void T01DriftChamberHit::operator delete(void* aHit)
{
    T01DriftChamberHitAllocator.FreeSingle((T01DriftChamberHit*)aHit);
}
# Sensitive Detector class

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

```cpp
#include "G4VSensitiveDetector.hh"
#include "T01DriftChamberHit.hh"
class G4Step;
class G4HCofThisEvent;
class T01DriftChamber : public G4VSensitiveDetector
{
  public:
    T01DriftChamber(G4String name);
    virtual ~T01DriftChamber();
    virtual void Initialize(G4HCofThisEvent*HCE);
    virtual G4bool ProcessHits(G4Step*aStep,
      G4TouchableHistory*ROhist);
    virtual void EndOfEvent(G4HCofThisEvent*HCE);
  private:
    T01DriftChamberHitsCollection * hitsCollection;
    G4int HCID;
};
```
Implementation of Sensitive Detector

T01DriftChamber::T01DriftChamber(G4String name)
:G4VSensitiveDetector(name)
{
    G4String HCname;
    collectionName.insert(HCname="driftChamberColl");
    HCID = -1;
}

void T01DriftChamber::Initialize(G4HCofThisEvent*HCE)
{
    hitsCollection = new T01DriftChamberHitsCollection
        (SensitiveDetectorName,collectionName[0]);
    if(HCID<0)
    {
        HCID = G4SDManager::GetSDMpointer() ->GetCollectionID(hitsCollection);
    }
    HCE->AddHitsCollection(HCID,hitsCollection);
}
Implementation of Sensitive Detector

G4bool T01DriftChamber::ProcessHits
  (G4Step*aStep,G4TouchableHistory*ROhist)
{
  T01DriftChamberHit* aHit =
    new T01DriftChamberHit();
  // some set methods
  hitsCollection->insert(aHit);
  return true;
}

void T01DriftChamber::EndOfEvent(G4HCofThisEvent*HCE)
{;;}
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 got 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
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();
G4ThreeVector worldPos = preStepPoint->GetPosition();
G4ThreeVector localPos = theTouchable->GetHistory()->GetTopTransform().TransformPoint(worldPos);
Readout geometry

- 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.

![Diagram showing the relationship between the tracking geometry and readout geometry](image)
Defining a sensitive detector

- Basic strategy
  
  ```cpp
  G4LogicalVolume* myLogCalor = ......;
  G4VSensitiveDetector* pSensitivePart =
      new MyCalorimeterSD("/mydet/calorimeter");
  G4SDManager* SDMan = G4SDManager::GetSDMpointer();
  SDMan->AddNewDetector(pSensitivePart);
  myLogCalor->SetSensitiveDetector(pSensitivePart);
  ```

- 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
  - One logical volume cannot have more than one detector objects. But, one detector object can generate more than one kinds of hits.
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.
Useage of G4HCofThisEvent

G4SDManager* SDman = G4SDManager::GetSDMpointer();
DHC1ID = Sdman
    ->GetCollectionID(colName="chamber1/driftChamberColl");
DHC2ID = Sdman
    ->GetCollectionID(colName="chamber2/driftChamberColl");

G4HCofThisEvent * HCE = evt->GetHCofThisEvent();
T01DriftChamberHitsCollection* DHC1 = 0;
T01DriftChamberHitsCollection* DHC2 = 0;
if(HCE)
{
    DHC1 = (T01DriftChamberHitsCollection*)(HCE->GetHC(DHC1ID));
    DHC2 = (T01DriftChamberHitsCollection*)(HCE->GetHC(DHC2ID));
}
Useage of G4HCofThisEvent

if(DHC1)
{
    int n_hit = DHC1->entries();
    G4cout << "Drift Chamber 1 has "
    << n_hit << " hits." << G4endl;
    for(int i1=0;i1<n_hit;i1++)
    {
        T01DriftChamberHit* aHit = (*DHC1)[i1];
        aHit->Print();
    }
}

- This scheme can also be utilized for Digitization.