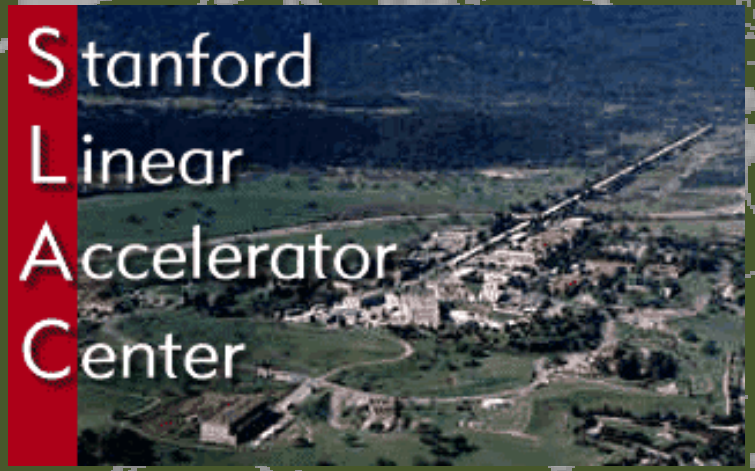


Detector Sensitivity

Makoto Asai (SLAC)

Geant4 Tutorial Course @ Bordeaux

November 2005

Geant4

Contents

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

- New : G4MultiFunctionalDetector, G4VPrimitiveSensitivity, G4VSDFilter and G4THitsMap
  - Convenient to medical and space applications
- Accumulating event data

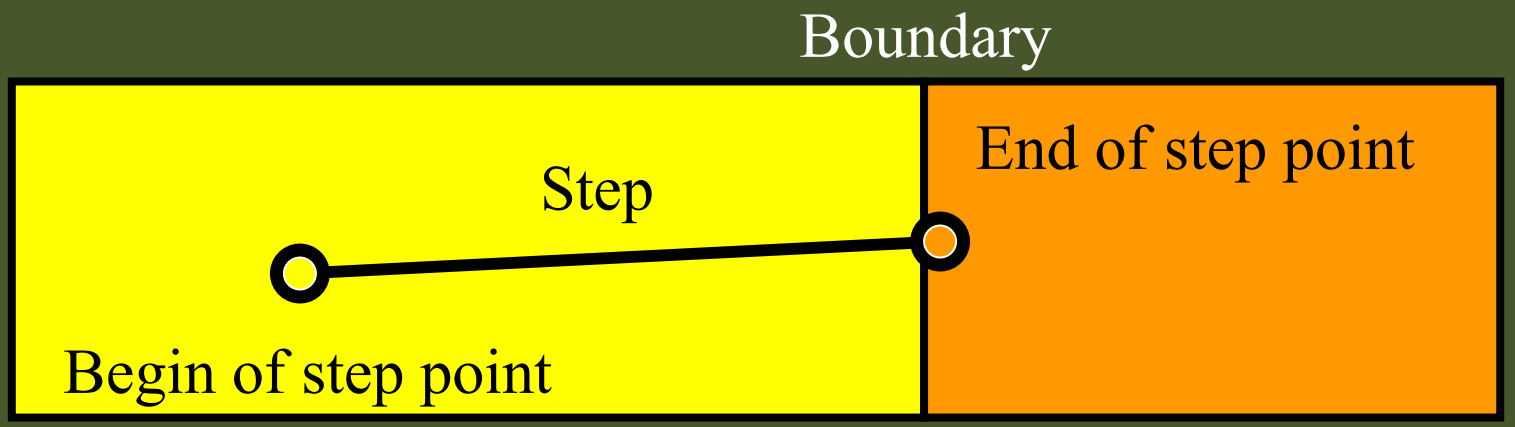
Sensitive detector and Hit

- 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”.



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 should be assigned to a volume and is automatically accessed at tracking time, the digitize() method of each G4VDigitizerModule must be explicitly invoked by the user's code (e.g. at EventAction).
  - Geant4 kernel will not invoke G4VDigitizerModule by default.

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 MyDriftChamberHit : public G4VHit
{
  public:
      MyDriftChamberHit(some_arguments);
      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.

```
#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;
};
```

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 drift chambers can have their dedicated sensitive detector objects. But, the functionalities of them are exactly the same to each other and thus they can share the same class. See examples/extended/analysis/A01 as an example.

Implementation of Sensitive Detector - 1

```
MyDriftChamber::MyDriftChamber(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 MyDriftChamber::Initialize(G4HCofThisEvent*HCE)
{
  if(collectionID<0) collectionID = GetCollectionID(0);
  hitsCollection = new MyDriftChamberHitsCollection
        (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 registered to G4SDManager. Thus, this method cannot be used 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 MyDriftChamber::ProcessHits
  (G4Step*aStep,G4TouchableHistory*ROhist)
{
  MyDriftChamberHit* aHit = new MyDriftChamberHit();
  ...
  // 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 MyDriftChamber::EndOfEvent(G4HCofThisEvent*HCE)
{;}
```

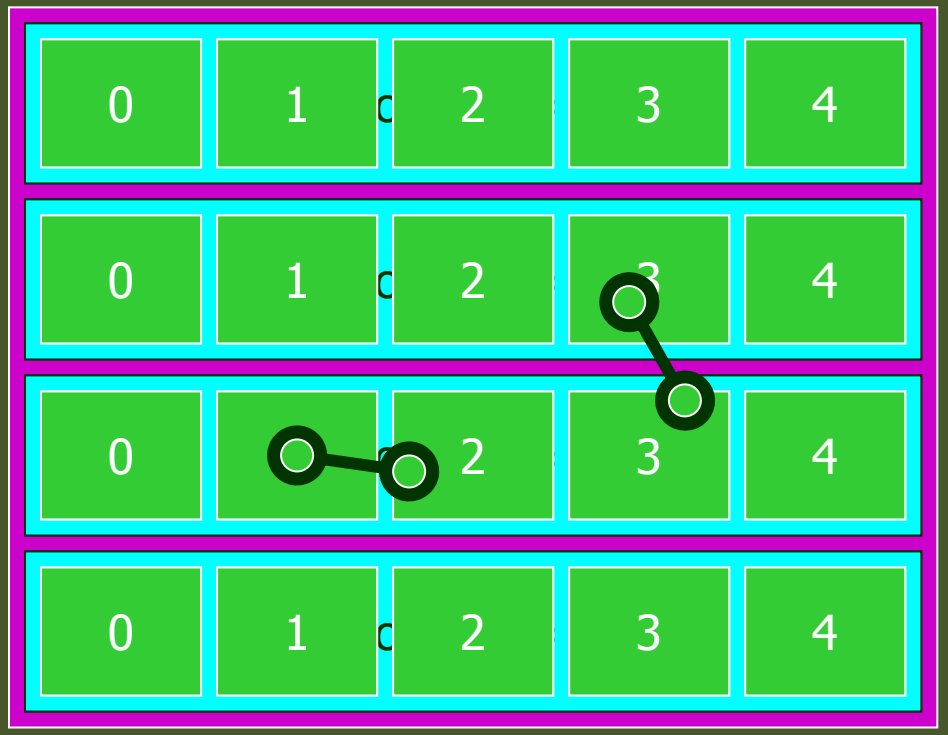
- This method is invoked at the end of processing an event.
  - It is invoked even if the event is aborted.
  - It is invoked before UserEndOfEventAction.

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 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
- 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 collect copy number for "PreStepPoint" if you directly access to the physical volume.
- Use touchable to get the proper copy number, transform matrix, etc.

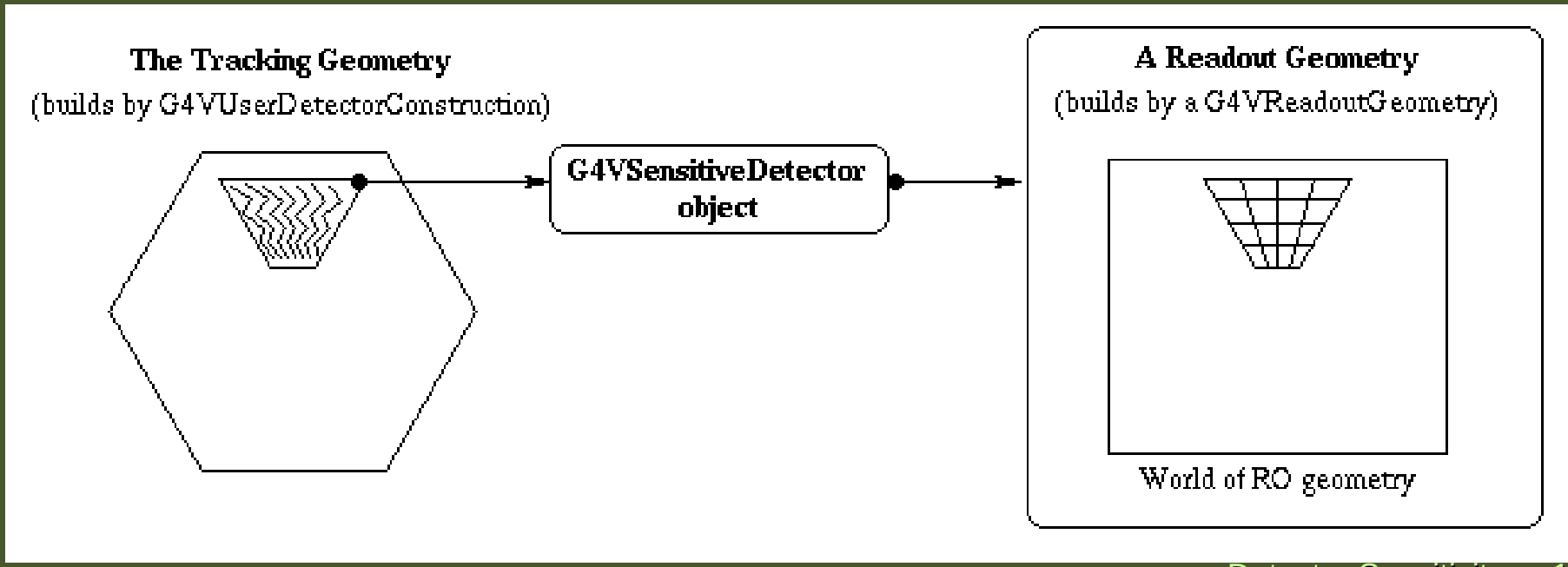
Touchable

- G4TouchableHistory has information of geometrical hierarchy of the point.

```
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);
```

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 = ……;
G4VSensetiveDetector* pSensetivePart =
   new MyCalorimeter(“/mydet/calorimeter1”);
G4SDManager* SDMan = G4SDManager::GetSDMpointer();
SDMan->AddNewDetector(pSensitivePart);
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) 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

```
static int CHCID = -1;
If(CHCID<0) 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(); }
}
```

cast

- This scheme can be adapted also for Digitization.

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(), 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.

Concrete sensitivity classes

- Up to the current version (7.1), Geant4 provides only an abstract base class (G4VSensitiveDetector) for the user to define his/her detector sensitivity.
  - Various example detector classes are provided.
    - This is enough for HEP experiments, since their interest is mostly storing hits in their detectors.
  - But not convenient for space and medical applications.
    - Their interest is mainly scoring dose/flux.
- We will introduce G4MultiFunctionalDetector (concrete class derived from G4VSensitiveDetector) that allows the user to register concrete class objects of G4VPrimitiveSensitivity to build a scoring detector of his/her needs.
  - G4PSEnergyDeposit, G4PSFlatSurfaceFlux, G4PSDoseDeposit, G4PSTrackLength, etc. (class names are still preliminary) will be provided.
  - We will continue working for additional primitive sensitivity concrete classes.

Concrete sensitivity classes

- Each G4VPrimitiveSensitivity class generates one hits collection per event. By registering more than one classes of G4VPrimitiveSensitivity, G4MultiFunctionalDetector generates more than one collections.
- New class G4VSDFilter will be introduced. It can be attached to G4VSensitiveDetector and/or G4VPrimitiveSensitivity to define which kinds of tracks are to be scored.
  - E.g., surface flux of protons of more than 1 GeV/c can be scored by G4PSFlatSurfaceFlux with a filter.
- Current G4Scorer and its related classes will become obsolete, but they will be kept with limited functionalities for a while for backward compatibility sake.

G4THitsMap

- G4THitsMap template class (an alternative to G4THitsCollection) will be introduced. It is also a derived class of G4VHitsCollection.
  - It is more convenient for scoring purposes. It does NOT mandate G4VHit concrete class to be stored, but for example a simple double value can be mapped with a copy number.
  - All new primitive sensitivity classes use G4THitsMap.

For example...

```
MyDetectorConstruction::Construct()
{ …  G4LogicalVolume* myCellLog =  new G4LogicalVolume(…);
     G4VPhysicalVolume* myCellPhys = new G4PVParametrised(…);
     G4MultiFunctionalDetector* myScorer = new G4MultiFunctionalDetector("myCellScorer");
     G4SDManager::GetSDMpointer()->AddNewDetector(myScorer);
     myCellLog->SetSensitiveDetector(myScorer);
     G4VPrimitiveSensitivity* totalSurfFlux = new G4PSFlatSurfaceFlux("TotalSurfFlux");
     myScorer->Register(totalSurfFlux);
     G4VPrimitiveSensitivity* protonSufFlux = new G4PSFlatSurfaceFlux("ProtonSurfFlux");
     G4VSDFilter* protonFilter = new G4SDParticleFilter("protonFilter");
     protonFilter->Add("proton");
     protonSurfFlux->SetFilter(protonFilter);
     myScorer->Register(protonSurfFlux);
     G4VPrimitiveSensitivity* totalDose = new G4PSDoseDeposit("TotalDose");
     myScorer->Register(totalDose);
}
```

No need of implementing sensitive detector !

Accumulating event data

- For scoring purposes, you need to accumulate a physical quantity (e.g. energy deposition of a step) for entire run of many events. In such a case, do NOT add up individual energy deposition of each step directly to a variable for entire run.
  - Compared to the total sum for entire run, energy deposition of single step is too tiny. Rounding error problem may easily happen.
    - Total energy deposition of 1 million events of 1 GeV incident particle ends up to 1 PeV (10^{15} eV), while energy deposition of each single step is O(1 keV) or even smaller.
- Create your own Run class derived from G4Run, and implement RecordEvent(const G4Event*) virtual method. Here you can get all output of the event so that you can accumulate the sum of an event to a variable for entire run.
  - RecordEvent(const G4Event*) is automatically invoked by G4RunManager.
  - Your run class object should be instantiated in GenerateRun() method of your UserRunAction.
- In particular with newly introducing G4VPrimitiveSensitivity concrete classes, this is the simplest and most widely applicable way.

Customized run class

```
#include "G4Run.hh"
#include "G4Event.hh"
#include "G4THitsMap.hh"
Class MyRun : public G4Run
{
  public:
    MyRun();
    virtual ~MyRun();
    virtual void RecordEvent(const G4Event*);
  private:
    G4int nEvent;
    G4int totalSurfFluxID, protonSurfFluxID, totalDoseID;
    G4THitsMap<G4double> totalSurfFlux;
    G4THitsMap<G4double> protonSurfFlux;
    G4THitsMap<G4double> totalDose;
    G4THitsMap<G4double>* eventTotalSurfFlux;
    G4THitsMap<G4double>* eventProtonSurfFlux;
    G4THitsMap<G4double>* eventTotalDose
  public:
    … access methods …
};
```

Note :

This sample code uses newly introducing concrete sensitivity classes.

Implement how you accumulate event data

Customized run class

```
MyRun::MyRun() : nEvent(0)
{
  G4SDManager* SDM = G4SDManager::GetSDMpointer();
  totalSurfFluxID = SDM->GetCollectionID("myCellScorer/TotalSurfFlux");
  protonSurfFluxID = SDM->GetCollectionID("myCellScorer/ProtonSurfFlux");
  totalDoseID = SDM->GetCollectionID("myCellScorer/TotalDose");
}
void MyRun::RecordEvent(const G4Event* evt)
{
  nEvent++;
  G4HCofThisEvent* HCE = evt->GetHCofThisEvent();
  eventTotalSurfFlux = (G4THitsMap<G4double>*)(HCE->GetHC(totalSurfFluxID));
  eventProtonSurfFlux = (G4THitsMap<G4double>*)(HCE->GetHC(protonSurfFluxID));
  eventTotalDose = (G4THitsMap<G4double>*)(HCE->GetHC(totalDose));
  totalSurfFlux += *eventTotalSurfFlux;
  protonSurfFlux += *eventProtonSurfFlux;
  totalDose += *eventTotalDose;
}
```

name of G4MultiFunctionalDetector object

name of G4VPrimitiveSensitivity object

No need of loops.
+= operator is provided !

RunAction with customized run

```
G4Run* MyRunAction::GenerateRun()
{ return (new MyRun()); }
void MyRunAction::EndOfRunAction(const G4Run* aRun)
{
  MyRun* theRun = (MyRun*)aRun;
  // … analyze / record / print-out your run summary
  // MyRun object has everything you need …
}
```

- As you have seen, to accumulate event data, you do NOT need
  - Event / tracking / stepping action classes
- All you need are your Run and RunAction classes.
- With newly introducing concrete sensitivity classes, you do NOT even need
  - Sensitive detector implementation

In summary

- SensitiveDetector is the class to be associated to G4LogicalVolume.
- In case you want to store hits of each event,
  - Implement hit, sensitive detector and event action classes.
- In case you want to accumulate event data and to get the run summary,
  - Use newly introduced concrete sensitivity classes and,
  - Implement run and run action classes.