IEEE Nuclear Science Symposium and Medical Imaging Conference
Short Course

Simulation Techniques Using Geant4

Maria Grazia Pia (*INFN Genova, Italy)*
MariaGrazia.Pia@ge.infn.it

Dresden, 18 October 2008

http://www.ge.infn.it/geant4/events/nss2008/geant4course.html

This course exploits training material developed by several Geant4 Collaboration members: thanks to all of them!

Overview

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

Extracting information from the simulation

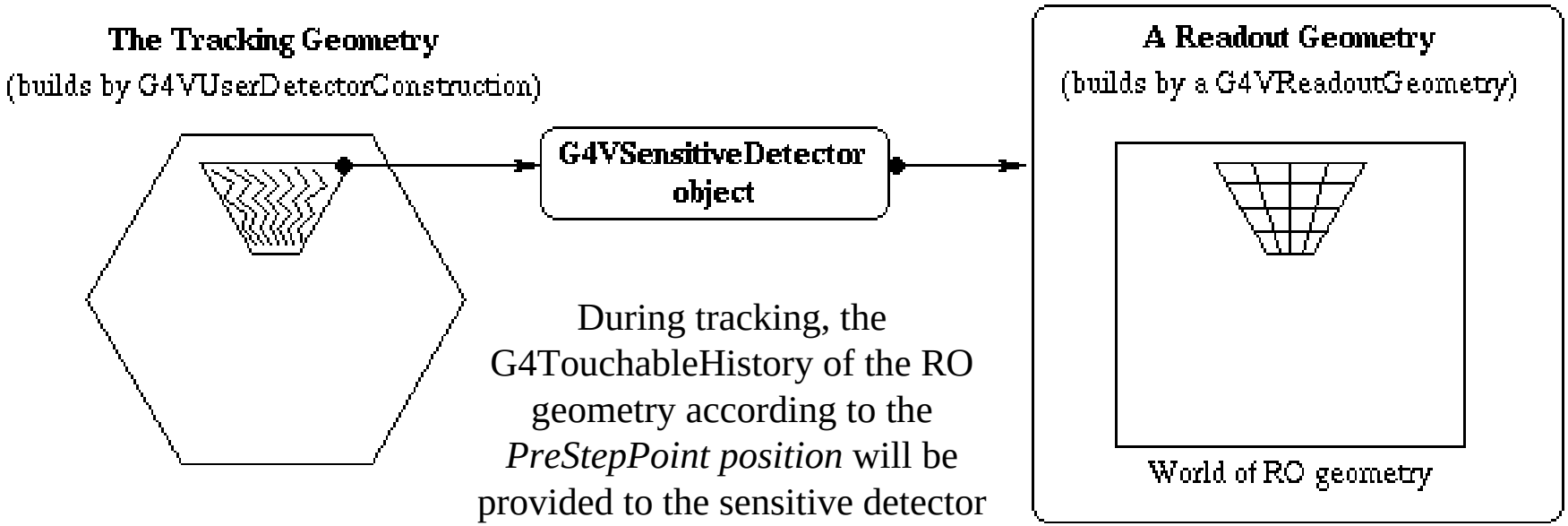
- Once the mandatory classes (Detector Construction, Physics List, Primary Generation) are implemented, the Geant4 application does not yet include functionality to extract information produced in the simulation

- A user must provide his/her own code to extract information relevant to the simulation application and describe the detector response

- Geant4 concepts for such functionality are
  - **Sensitive Detector** (optional with **readout geometry**)
  - **Hits** and **Hits collections**
  - **Digis** and **Digis collections**

Readout geometry

- Readout geometry is a virtual and artificial geometry which can be defined in parallel to the real detector geometry
- Tracks will be traced in the “real” geometry, but the sensitive detector can have its own geometry for readout purposes, e.g. to find the cell the current hit belongs to
- The 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 the readout geometry



How to create a Readout Geometry

- Derive your own concrete class from the **G4VReadoutGeometry** abstract base class
- The geometry setup is done in the **same way** as for the “real” tracking geometry:
  - Create solids, logical and physical volumes
- However, the materials used in the RO geometry are **dummy materials**
  - i.e. they are not used
- A **sensitive detector** for the RO geometry must be defined but is not used!
  - This means that you need to declare the sensitive parts of the RO geometry by setting a non NULL sensitive detector pointer to the logical volume

Touchable

- Each G4StepPoint object has
  - Position in world coordinate system
  - Global and local time
  - Material
  - G4TouchableHistory for geometrical information

- G4TouchableHistory object provides information on the geometrical hierarchy
  - copy number
  - transformation / rotation to its mother

Sensitive detector

- A logical volume becomes **sensitive** if it has a pointer to a sensitive detector

- A sensitive detector **can be instantiated several times**, where the instances are assigned to different logical volumes
  - SD objects must have unique detector names
  - A **logical volume can only have one SD object** attached
  - But you can implement your detector to have multiple functionality

- Two possibilities to make use of the SD functionality
  - **Create your own sensitive detector**
    - Highly customizable
  - **Use Geant4 built-in tools**: Primitive Scorers

Creating your own sensitive detector

- A powerful way of extracting information from the physics simulation is to define your own SD
- The ingredients of the scoring setup are:

	Concrete Class	Base Class
Sensitive Detector	*MySensitiveDetector*	G4VSensitiveDetector
Readout Geometry	*MyReadoutGeometry* (optional)	G4VReadoutGeometry
Hit	*MyHit*	G4VHit
		Template Class
Hits Collection		G4THitsCollection<your hit class>

- Derive your own concrete classes from the base classes and customize them according to your needs

Basic strategy to retrieve information - 1

- Assume, you have already created the detector geometry
  - Shape and size (Solid) of your detector, Material
  - Logical volumes
  - Physical volumes
- Implement a sensitive detector and assign an instance of it to the *logical volume* of your detector geometry setup
  - Then this volume becomes sensitive
  - The sensitive detector will become “active” for each particle step, if the step starts inside this logical volume
- Optionally: Implement a readout geometry and attach it to the sensitive detector

Basic strategy to retrieve information - 2

- Then, create **hit** objects in your sensitive detector using information from particle steps
- **Hit** is a snapshot of the physical interaction of a track or an accumulation of interactions of tracks in the sensitive or interesting region of your detector
  - *You need to create hit class(es) according to your needs*
- Use **Touchable** of the **Readout Geometry** to retrieve geometrical information associated with hits
- Store your hits in **hit collections**
  - hit collections are automatically associated to the G4Event object
- Finally, process the information associated with hits in user action classes (G4UserEventAction,G4UserRunAction) to obtain a summary of the relevant event/run features

Using built-in scorers

- Alternatively, you can use a predefined sensitive detector **G4MultiFunctionalDetector** and primitive scorers:

	Concrete Class(es)
Sensitive Detector	G4MultiFunctionalDetector
Primitive Scorers	G4PSEnergyDeposit, G4PSTracklength, ...
	Template Class
Hits Collection	G4THitsMap<G4double>

  - Each primitive scorer stores **one physics quantity** for each ***physical volume*** (accumulated over an event)
    - Many scorers are provided by Geant4 (energy deposit, flux, ...)

Basic strategy to retrieve information

- Assume, you have already created the detector geometry
  - Shape and size (Solid) of your detector, Material
  - Logical volumes
  - Physical volumes
- **Assign** an instance of the Geant4 **multifunctional detector** (G4MultiFunctionalDetector) to the ***logical volume*** of your detector geometry set-up
- **Register** instances of the required **primitive scorers** to your **multifunctional detector**
- Finally, **process** the content of **hit maps**

Sensitive Detector and Hits

- Using information from particle steps, a sensitive detector either
  - constructs one or more hits objects or
  - accumulates values to existing hits
    - g.g.: a tracker detector stores position, time of hit etc. a calorimeter stores energy deposit etc.
- Hit objects can be supplied information in the ProcessHits method of the concrete class of the SD
  - This function has pointers to the current G4Step object and the G4TouchableHistory of the RO geometry (if defined) as arguments
  - Note that you must get the volume information from the "PreStepPoint"

Hit

- Hit is a **user-defined** class, which derives from the base class G4VHit

- You can store various kind of information by implementing your own concrete Hit class
- Typically one records quantities like:
  - Position and time of the step
  - Momentum and energy of the track
  - Energy deposit in the step
  - Geometrical information
  - etc.

Example of a Hit class

```
// header file: MyHit.hh
#include “G4VHit.hh“
class MyHit : public G4VHit {
public:
MyHit();
virtual ~MyHit();
...
inline void SetEnergyDeposit(G4double energy) { energyDeposit = energy; }
inline G4double GetEnergyDeposit() { return energyDeposit;}
... // more member functions

private:
G4double energyDeposit;
... // more data members
};
```

Hits Collection

- Once created in the sensitive detector, instances of the concrete hit class must be stored in a dedicated collection:
  - Template class **G4THitsCollection**
  - the template parameter is the concrete hit class: G4THitsCollection<MyHit>
- Hits collections can be accessed in various phases of the simulation
  - At the end of each event through the G4Event object
    - to analyse the event and store useful information
  - During event processing through the sensitive detector manager, G4SDManager
    - to perform event filtering

Digis

- A Hit is created when a particle step is inside a detector
- In contrast, a Digit represents the output of a detector:
  - e.g. ADC/TDC count, trigger signal, ...
- It is created from the information from hits and/or other digits
- Similarly as for hits, you need to implement customized digit class(es) of your own
- The base clase of digits is G4VDigi
- Digit objects are stored in digit collections:
  - Template class G4TDigiCollection
- G4Event has a GDCofThisEvent object, that is a container for digit collections
- Note the similarity with Hits

Digitization

- Digits are created in a **digitizer module**
- Create your digitizer module by inheriting from the base class **G4VDigitizerModule**
  - Overload the pure virtual method Digitize(): Create digit objects in this function and store them in digit collections
- NOTE: In contrast to the ProcessHits() function (which creates and stores hits), the Digitize() method is **NOT automatically called** by the Geant4 kernel:
  - You should invoke this function yourself explicitly
    - e.g. in the user action classes