

# Interaction with the Geant4 kernel – part 2

---

Luciano Pandola

INFN – Laboratori Nazionali del Sud

SFB & GSSI Geant4 School, Munich, April 16<sup>th</sup>-20<sup>th</sup>, 2018



# Part I: Sensitive Detectors

---



# Sensitive Detector (SD)

---

- A **logical volume** becomes **sensitive** if it has a pointer to a **sensitive detector** (`G4VSensitiveDetector`)
  - A sensitive detector can be instantiated **several times**, where the instances are assigned to **different logical volumes**
    - Note that 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 many functionalities)
- **Two possibilities** to make use of the SD functionality:
  - Create **your own sensitive detector** (using class inheritance)
    - Highly **customizable**
  - Use Geant4 **built-in tools**: Primitive scorers

# Adding sensitivity to a logical volume

- Create an instance of a sensitive detector and register it to the SensitiveDetector Manager
- Assign the pointer of your SD to the logical volume of your detector geometry
- Must be done in `ConstructSDandField()` of the user geometry class

```
G4VSensitiveDetector* mySensitive  
= new MySensitiveDetector(SDname="/MyDetector");
```

} create instance

```
G4SDManager* sdMan =G4SDManager::GetSDMpointer();  
sdMan->AddNewDetector(mySensitive);
```

} Register to the SD manager

```
SetSensitiveDetector("LVname",mySensitive);
```

} assign to logical volume

→  
Name of the logical volume

# Adding sensitivity to a logical volume - variant

- Create an instance of a sensitive detector and register it to the SensitiveDetector Manager
- Assign the pointer of your SD to the logical volume of your detector geometry
- Must be done in `ConstructSDandField()` of the user geometry class

```
G4VSensitiveDetector* mySensitive  
    = new MySensitiveDetector(SDname="/MyDetector");
```

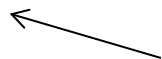
} create instance

```
G4SDManager* sdMan =G4SDManager::GetSDMpointer();  
sdMan->AddNewDetector(mySensitive);
```

} Register to the SD manager

```
logicVol->SetSensitiveDetector(mySensitive);
```

} assign to logical volume



Pointer of the logical volume

# Part II: Native Geant4 scoring



---



# Extract useful information

---

- Geant4 provides a number of **primitive scorers**, each one **accumulating one physics quantity** (e.g. total dose) for an event
- This is alternative to the **customized sensitive detectors** (see later in lecture #3), which can be used with full flexibility to gain complete control
- It is **convenient** to use primitive scorers **instead** of **user-defined sensitive detectors** when:
  - you are not interested in recording each individual step, but **accumulating physical quantities** for an event or a run
  - you have **not too many** scorers



# G4MultiFunctionalDetector

---

- **G4MultiFunctionalDetector** is a concrete class derived from **G4VSensitiveDetector**
- It should be **assigned to a logical volume** as a **kind of (ready-for-the-use) sensitive detector**
- It takes an **arbitrary number** of **G4VPrimitiveScorer** classes, to define the **scoring quantities** that you need
  - Each **G4VPrimitiveScorer** accumulates **one physics quantity** for each physical volume
  - E.g. **G4PSDoseScorer** (a concrete class of **G4VPrimitiveScorer** provided by Geant4) **accumulates dose** for each cell
- By using this approach, **no need to implement sensitive detector** and **hit classes!**





# G4VPrimitiveScorer

---

- Primitive **scorers** (classes derived from **G4VPrimitiveScorer**) have to be registered to the **G4MultiFunctionalDetector**
  - ->**RegisterPrimitive()**,
  - ->**RemovePrimitive()**
- They are designed to **score one kind of quantity** (surface flux, total dose) and to **generate one hit collection** per event
  - automatically named as  
**<MultiFunctionalDetectorName>/<PrimitiveScorerName>**
    - **hit collections** can be **retrieved** in the **EventAction** or **RunAction** (as those generated by sensitive detectors)
    - **do not share** the same **primitive scorer object** among **multiple G4MultiFunctionalDetector** objects (results may **mix up!**)
      - Create as many instances of the scorer as needed

myCellScorer/TotalSurfFlux  
myCellScorer/TotalDose

# For example ...

```
MyDetectorConstruction::ConstructSDandField()
```

```
{
```

```
G4MultiFunctionalDetector* myScorer = new  
G4MultiFunctionalDetector("myCellScorer");
```

instantiate multi-  
functional detector

```
myCellLog->SetSensitiveDetector(myScorer);
```

attach to volume

```
G4VPrimitiveScorer* totalSurfFlux = new  
G4PSFlatSurfaceFlux("TotalSurfFlux");  
myScorer->RegisterPrimitive(totalSurfFlux);
```

create a primitive  
scorer (surface  
flux) and register  
it

```
G4VPrimitiveScorer* totalDose = new  
G4PSDoseDeposit("TotalDose");  
myScorer->RegisterPrimitive(totalDose);
```

create a primitive  
scorer (total dose)  
and register it

```
}
```



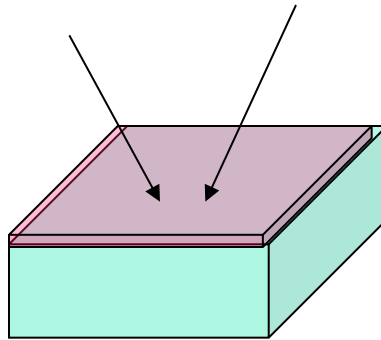
# Some primitive scorers that you may find useful

---

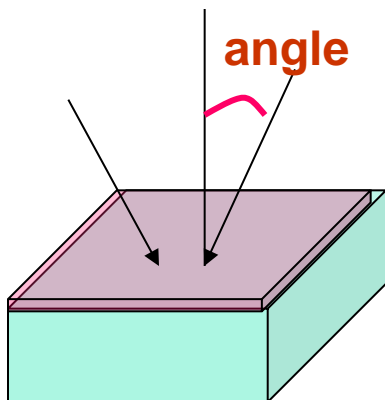
- Concrete Primitive Scorers (→ Application Developers Guide 4.4.5)
  - Track length
    - G4PSTrackLength, G4PSPassageTrackLength
  - Deposited energy
    - G4PSEnergyDeposit, G4PSDoseDeposit
  - Current/Flux
    - G4PSFlatSurfaceCurrent, G4PSSphereSurfaceCurrent, G4PSPassageCurrent, G4PSFlatSurfaceFlux, G4PSCellFlux, G4PSPassageCellFlux
  - Others
    - G4PSMinKinEAtGeneration, G4PSNofSecondary, G4PSNofStep, G4PSCellCharge

# A closer look at some scorers

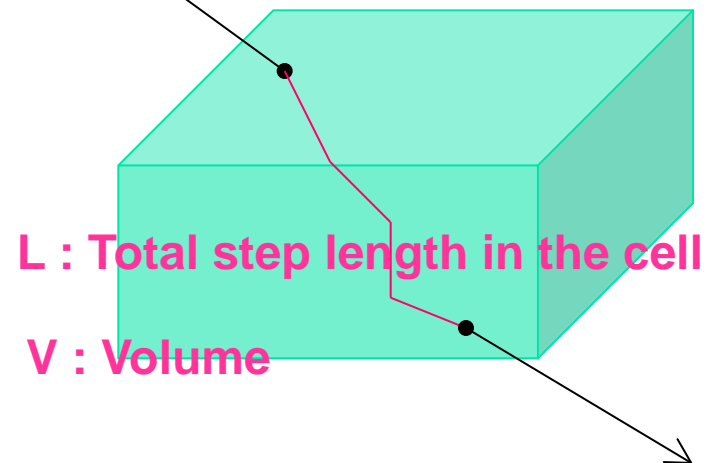
SurfaceCurrent :  
Count number of  
injecting particles  
at defined surface.



CellFlux :  
Sum of  $L / V$  of  
injecting particles  
in the geometrical  
cell.



SurfaceFlux :  
Sum up  
 $1/\cos(\text{angle})$  of  
injecting particles  
at defined surface



**L** : Total step length in the cell  
**V** : Volume

**V** : Volume



# G4VSDFilter

---

- A **G4VSDFilter** can be **attached** to **G4VPrimitiveScorer** to define **which kind of tracks** have to be scored (e.g. one wants to know surface flux of **protons only**)
  - **G4SDChargeFilter** (accepts only **charged** particles)
  - **G4SDNeutralFilter** (accepts only **neutral** particles)
  - **G4SDKineticEnergyFilter** (accepts tracks in a defined range of **kinetic energy**)
  - **G4SDParticleFilter** (accepts tracks of a **given particle type**)
  - **G4VSDFilter** (base class to create user-customized filters)



# For example ...

---

```
MyDetectorConstruction::ConstructSDandField()
```

```
{  
    G4VPrimitiveScorer* protonSurfFlux  
    = new G4PSFlatSurfaceFlux("pSurfFlux");  
    G4VSDFilter* protonFilter = new  
        G4SDParticleFilter("protonFilter");  
    protonFilter->Add("proton");  
  
    protonSurfFlux->SetFilter(protonFilter);  
  
    myScorer->RegisterPrimitive(protonSurfFlux);  
}
```

} create a primitive  
scorer (**surface  
flux**), as before

} create a **particle  
filter** and add  
**protons** to it

} **register** the **filter**  
to the primitive  
scorer

**register** the **scorer** to the  
multifunc detector (as  
shown before)

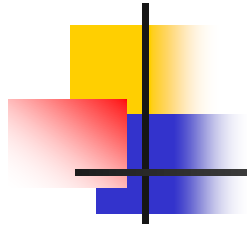
# How to retrieve information - part 1



---

- At the end of the day, one wants to **retrieve** the information from the scorers
  - **True** also for the **customized** hits collection
- Each scorer **creates a hit collection**, which is **attached** to the **G4Event** object
  - Can be retrieved and read at the **end of the event**, using an integer ID
  - Hits collections mapped as **G4THitsMap<G4double>\*** so can loop on the individual entries
  - **Operator += provided** which automatically sums up all hits (no need to loop manually)

# How to retrieve information - part 2



	Scorer 1	Scorer 2	HCoFThisEvent
Event#1	(0, 5.32)	(0, 1.43) (2, 7.41)	
Event#2	(1, 1.12)	(0, 1.11)	
Event#3	<i>empty</i>	<i>empty</i>	
...	...	...	
Event#N	(0, 7.12) (1, 1.15)	(0, 2.0)	

Annotations for Event#1:

- copyNb (key) points to the first element of the tuple (0).
- Quantity to be scored by the scorer (e.g. energy) points to the second element of the tuple (5.32).
- HCoFThisEvent points to the second tuple (2, 7.41).



# How to retrieve information – recipe

```
//needed only once
```

```
G4int collID = G4SDManager::GetSDMpointer()
```

```
->GetCollectionID("myCellScorer/TotalSurfFlux");
```

Get **ID** for the collection (given the name)

```
G4HCofThisEvent* HCE = event->GetHCofThisEvent();
```

Get **all HC** available in this event

```
G4THitsMap<G4double>* evtMap =
```

```
static_cast<G4THitsMap<G4double>*>
```

```
(HCE->GetHC(collID));
```

Get the HC with the **given ID** (need a cast)

```
for (auto pair : *(evtMap->GetMap())) {
```

```
G4double flux = *(pair.second);
```

```
G4int copyNb = *(pair.first);
```

```
}
```

**Loop** over the **individual entries** of the HC: the key of the map is the copyNb, the other field is the real content



# Hands-on session

---

- Task4
  - Task4c: Native scoring
- <http://geant4.lngs.infn.it/munich2018/task4>