The Sensitive Detector
A Native scoring

Geant4 School
at the XIV Seminar on software for nuclear, subnuclear and applied Physics
June 4th - 9th, 2017

Sensitive Detector I

The HIT concept

A hit is **a snapshot of the physical interaction** of a track **in the sensitive region** of a detector. In it you can store information associated with a `G4Step` object.
This information can be:

the position and time of the step,
the momentum and energy of the track,
the energy deposition of the step,
geometrical information,

The Sensitive Detector concept

`G4VSensitiveDetector` is an abstract base class which **represents a detector.** The principal mandate of a sensitive detector **is the construction of hit** objects using information from steps along a particle track.

The `ProcessHits()` method of `G4VSensitiveDetector` performs this task using `G4Step` objects as input.

Sensitive Detector II

`G4VSensitiveDetector` has three major virtual methods

`ProcessHits()`
is invoked by `G4SteppingManager` **when a step is composed in the G4LogicalVolume** which has the pointer to this sensitive detector.
The first argument of this method is a `G4Step` object of the current step.

`Initialize()`
This method is invoked at the beginning of each event.
The argument of this method is an object of the `G4HCofThisEvent` class:
Hit collections, where hits produced in this particular event are stored.

`EndOfEvent()`
This method is invoked at the end of each event.
The argument of this method is the same object as the previous method.

Sensitive Detector III

- 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**

  - The SD objects must have unique detector name
  - A logical volume can **only** have **one SD object attached**, but the detector can have many functionalities

- **Two possibilities** to make use of the SD functionality:

  Create **your own sensitive detector** (using class inheritance)
  ==> highly customisable

  Use the geant4 **built-in tools**: **the primitive scorers**

Make 'sensitive' 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 the **`ConstructSDandField()`** of the User geometry class

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

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

boxLogical->SetSensitiveDetector(mySensitive);
(or)
SetSensitiveDetector("LVname",mySensitive);
```

← Create instance

← Assign to logical volume

← Assign to logical volume (alternative)

Native Geant4 scoring

Extract useful information

Geant4 provides a numbers of primitive scorers, each of one accumulating one physics quantity (e.g. total dose) for an event

This is an alternative to the customised sensitive detectors (see later), which can be used with full flexibility to gain a complete control

It is convenient to use primitive scorers instead of the user-defined sensitive detectors when:

you are not interested in recording each individual step, but into accumulate physical quantities in an **event** or **run**

you have not too many scores

The G4MultifunctionalDetector

- **`G4MultifunctionalDetector`** is a concrete class derived by **`G4VSensitiveDetector`**
- It should be assigned to a logical volume as a kind of ready-to-use sensitive detector
- It takes an arbitrary number of **`G4VPrimitiveScorer`** classes, to define the scoring quantities you need
  - Each **`G4VPrimitiveScorer`** accumulates one physics quantity for each physical volume
  - e.g. G 4PSDoseScorer (a concrete class of G4VPrimitiveScorer) accumulates dose in each cell
- By using this approach: **no need to implement the Sensitive detector or the Hit class**

G4VPrimitiveScorer

- Primitive scores (classes derived from the **`G4VPrimitiveScorer`**) have to be registered to the **`G4MultiFunctionalDetector`**
  - ->RegisterPrimitive(); ->RemovePrimitive()
- They are defined **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

…. for example


```
MyDetectorConstruction::ConstructSDandField()
{
   G4MultiFunctionalDetector* myScorer = new
   G4MultiFunctionalDetector("myCellScorer");
   myCellLog->SetSensitiveDetector(myScorer);

   G4VPrimitiveSensitivity* totalSurfFlux = new
      G4PSFlatSurfaceFlux("TotalSurfFlux");
   myScorer->RegisterPrimitive(totalSurfFlux);
   G4VPrimitiveSensitivity* totalDose = new
      G4PSDoseDeposit("TotalDose");
   myScorer->RegisterPrimitive(totalDose);
}
```

instantiate multi-functional detector

attach to volume

create a primitive scorer (surface flux) and register it

create a primitive scorer (total dose) and register it

…. for example


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

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

instantiate multi-functional detector

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

attach to volume

```
G4VPrimitiveSensitivity* totalSurfFlux = new
   G4PSFlatSurfaceFlux("TotalSurfFlux");
myScorer->RegisterPrimitive(totalSurfFlux);
```

create a primitive scorer (surface flux) and register it

```
G4VPrimitiveSensitivity* totalDose = new
   G4PSDoseDeposit("TotalDose");
myScorer->RegisterPrimitive(totalDose);
```

create a primitive scorer (total dose) and register it

```
}
```

…. for example


```
MyDetectorConstruction::ConstructSDandField()
{
   G4MultiFunctionalDetector* myScorer = new
   G4MultiFunctionalDetector("myCellScorer");
```

instantiate multi-functional detector

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

attach to volume

```
   G4VPrimitiveSensitivity* totalSurfFlux = new
      G4PSFlatSurfaceFlux("TotalSurfFlux");
   myScorer->RegisterPrimitive(totalSurfFlux);
   G4VPrimitiveSensitivity* totalDose = new
      G4PSDoseDeposit("TotalDose");
   myScorer->RegisterPrimitive(totalDose);
}
```

create a primitive scorer (surface flux) and register it

create a primitive scorer (total dose) and register it

…. for example


```
MyDetectorConstruction::ConstructSDandField()
{
   G4MultiFunctionalDetector* myScorer = new
   G4MultiFunctionalDetector("myCellScorer");
   myCellLog->SetSensitiveDetector(myScorer);

   G4VPrimitiveSensitivity* totalSurfFlux = new
      G4PSFlatSurfaceFlux("TotalSurfFlux");
   myScorer->RegisterPrimitive(totalSurfFlux);
   G4VPrimitiveSensitivity* totalDose = new
      G4PSDoseDeposit("TotalDose");
   myScorer->RegisterPrimitive(totalDose);
}
```

instantiate multi-functional detector

attach to volume

create a primitive scorer (surface flux) and register it

create a primitive scorer (total dose) and register it

…. for example


```
MyDetectorConstruction::ConstructSDandField()
{
   G4MultiFunctionalDetector* myScorer = new
   G4MultiFunctionalDetector("myCellScorer");
   myCellLog->SetSensitiveDetector(myScorer);

   G4VPrimitiveSensitivity* totalSurfFlux = new
      G4PSFlatSurfaceFlux("TotalSurfFlux");
   myScorer->RegisterPrimitive(totalSurfFlux);
   G4VPrimitiveSensitivity* totalDose = new
      G4PSDoseDeposit("TotalDose");
   myScorer->RegisterPrimitive(totalDose);
}
```

instantiate multi-functional detector

attach to volume

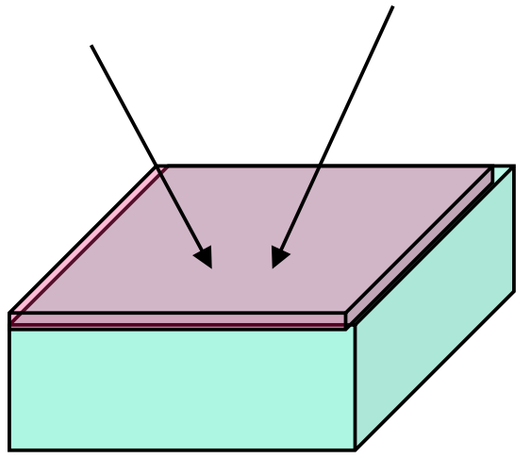
create a primitive scorer (surface flux) and register it

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
  - Deposited energy
    - G4PSEnergyDeposit, G4PSDoseDeposit
  - Current/Flux
    - G4PSFlatSurfaceCurrent, G4PSSphereSurfaceCurrent
  - Others
    - G4PSMinKinEAtGeneration, G4PSNofSecondary

A closer look at some scorer

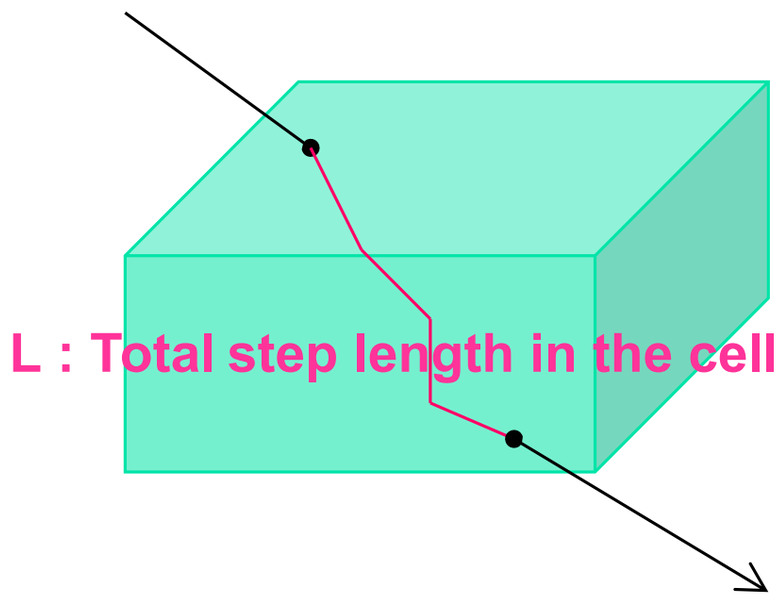


SurfaceCurrent :

Count number of injecting particles at defined surface.

CellFlux :

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



G4VSDFilter

- A **`G4VSDFilter`** can be attached to **`G4VPrimitiveSensitivity`** to define which kind of track 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)

… for example


```
MyDetectorConstruction::ConstructSDandField()
{
   G4VPrimitiveSensitivity* 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)

… for example


```
MyDetectorConstruction::ConstructSDandField()
{
    G4VPrimitiveSensitivity* 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)

… for example


```
MyDetectorConstruction::ConstructSDandField()
{
   G4VPrimitiveSensitivity* 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)

… for example


```
MyDetectorConstruction::ConstructSDandField()
{
   G4VPrimitiveSensitivity* 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 - I

- 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 hits (no need to loop)

How to retrieve information - II


```
//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)

```
 std::map<G4int,G4double*>::iterator itr;
 for (itr = evtMap->GetMap()->begin(); itr !=
       evtMap->GetMap()->end(); itr++) {
    G4double flux = *(itr->second);
    G4int copyNb  = *(itr->first);
}
```

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

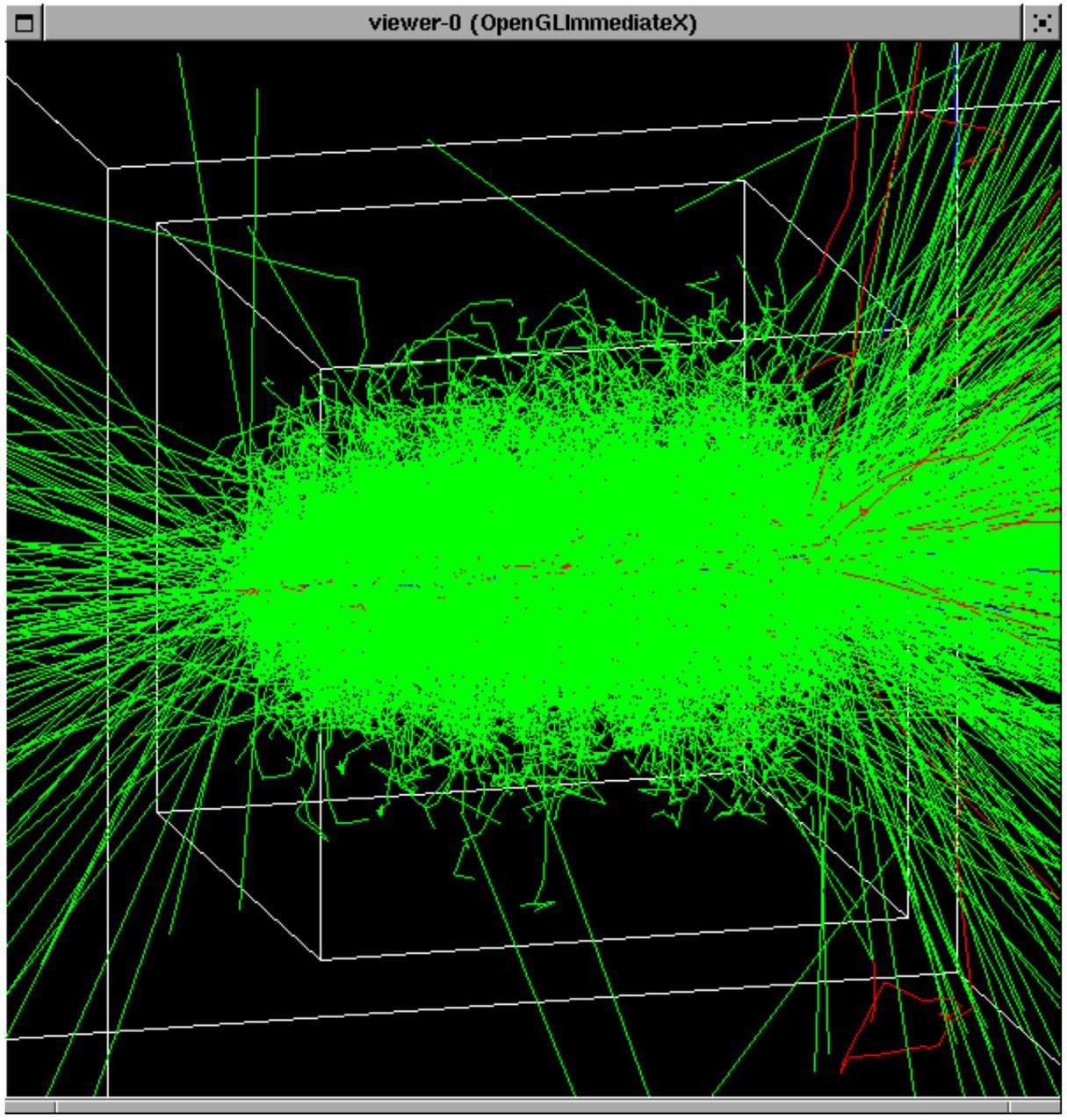
Command-based scoring

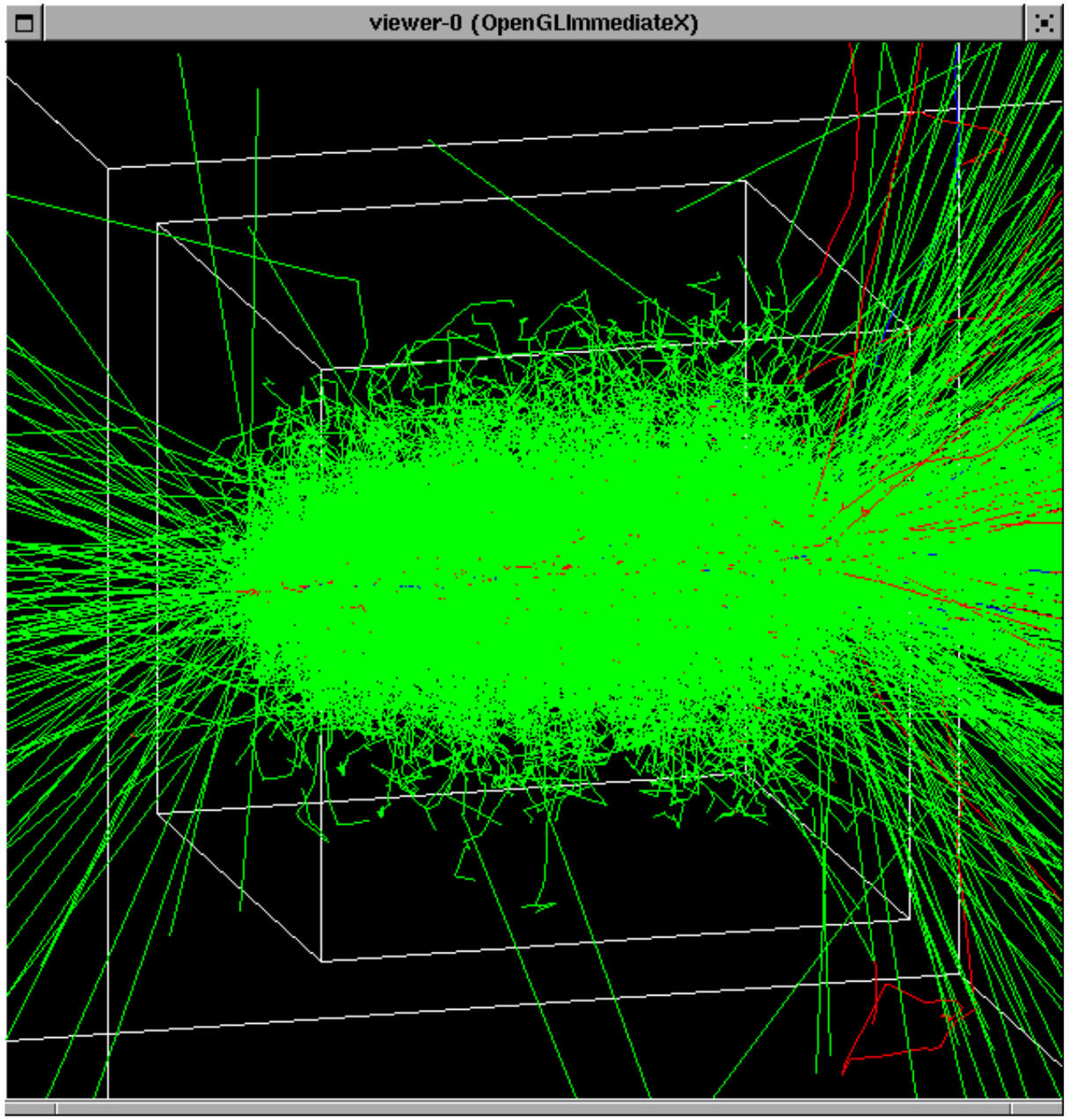
Under development!

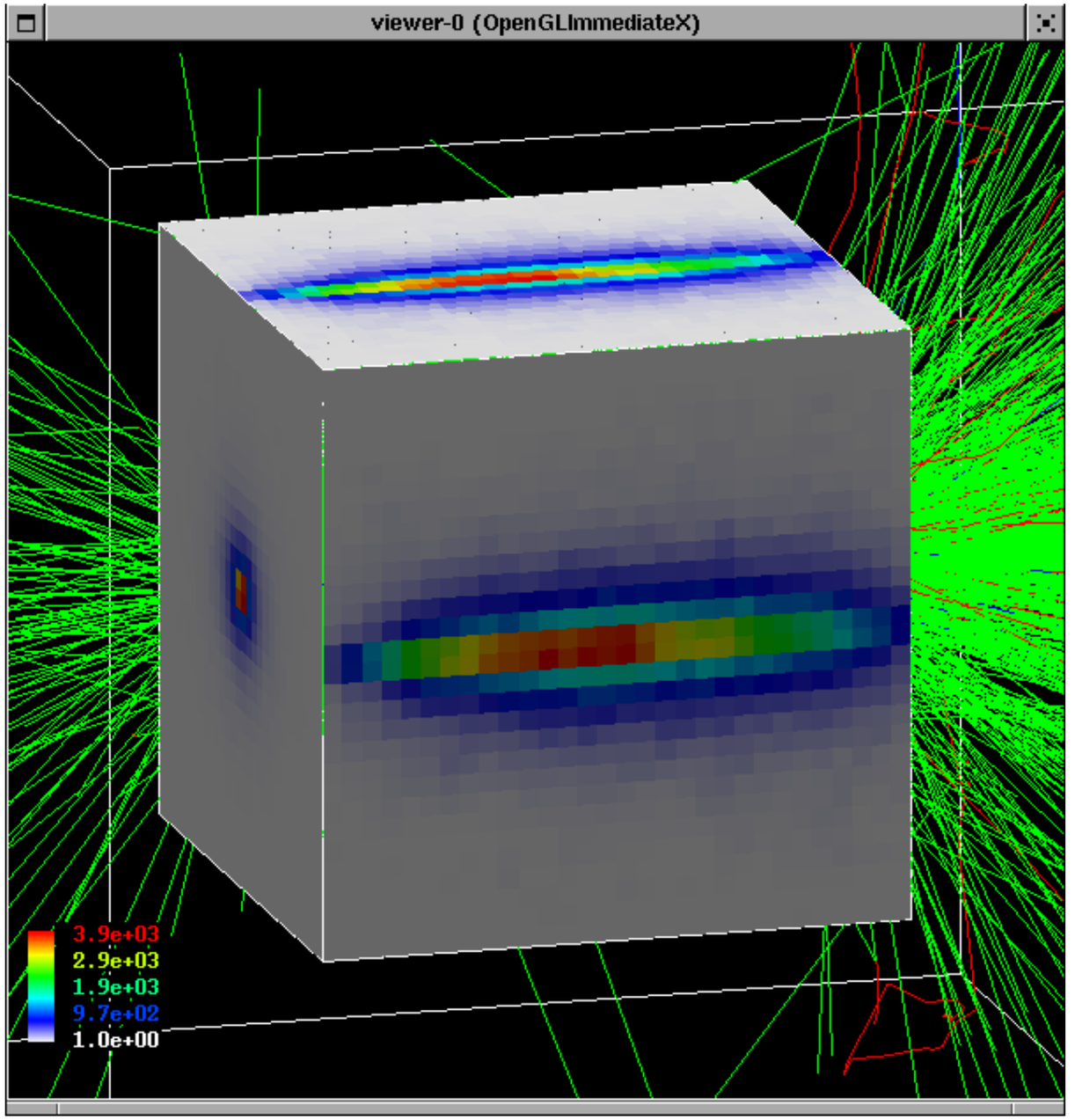
Thanks to the newly developed parallel navigation, an **arbitrary scoring mesh geometry** can be defined which is independent to the volumes in the mass geometry.
Also, G4MultiFunctionalDetector and primitive scorer classes now offer the built-in scoring of most-common quantities

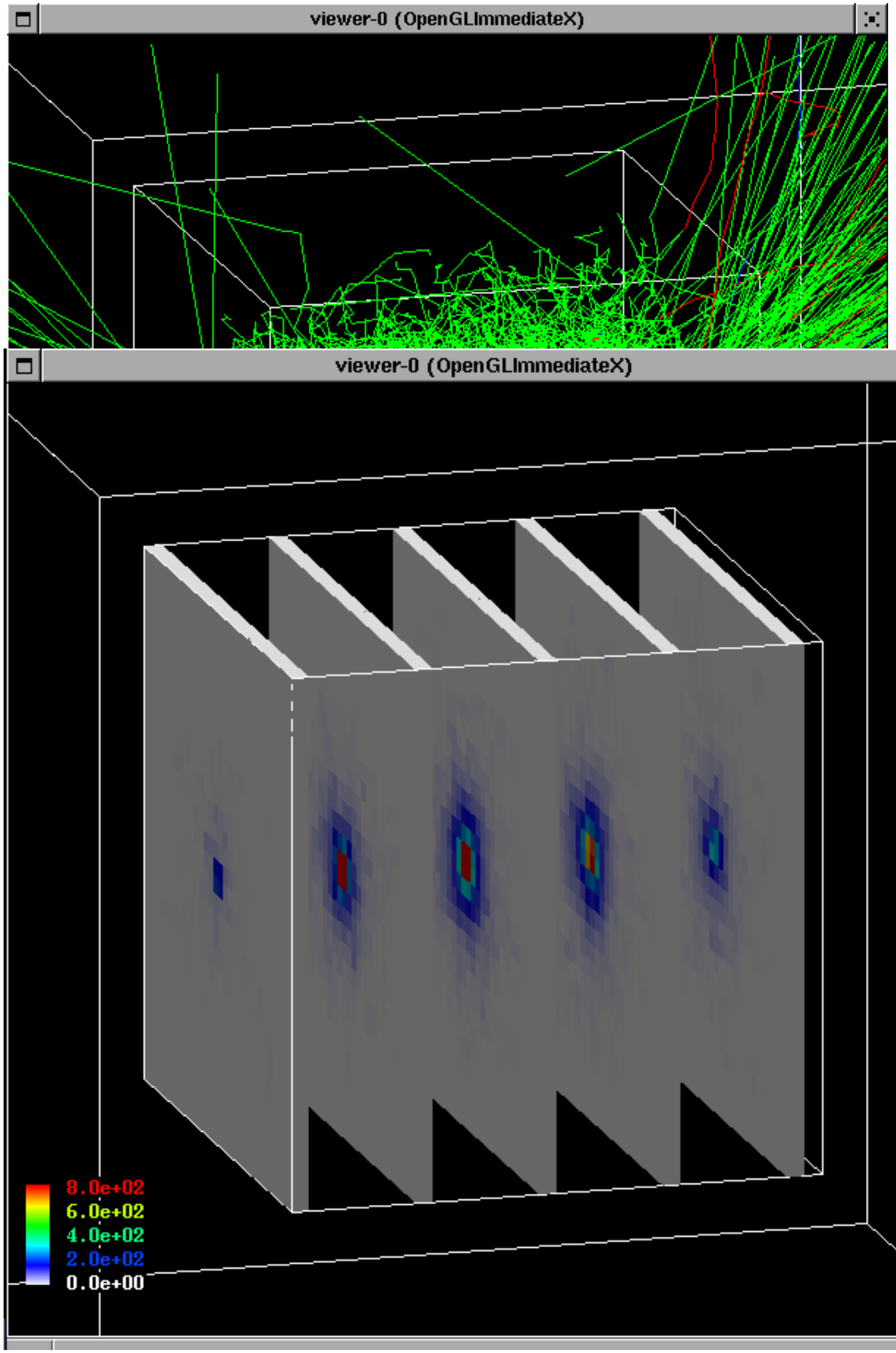
UI commands for scoring → no C++ required, apart from instantiating `G4ScoringManager` in `main()`

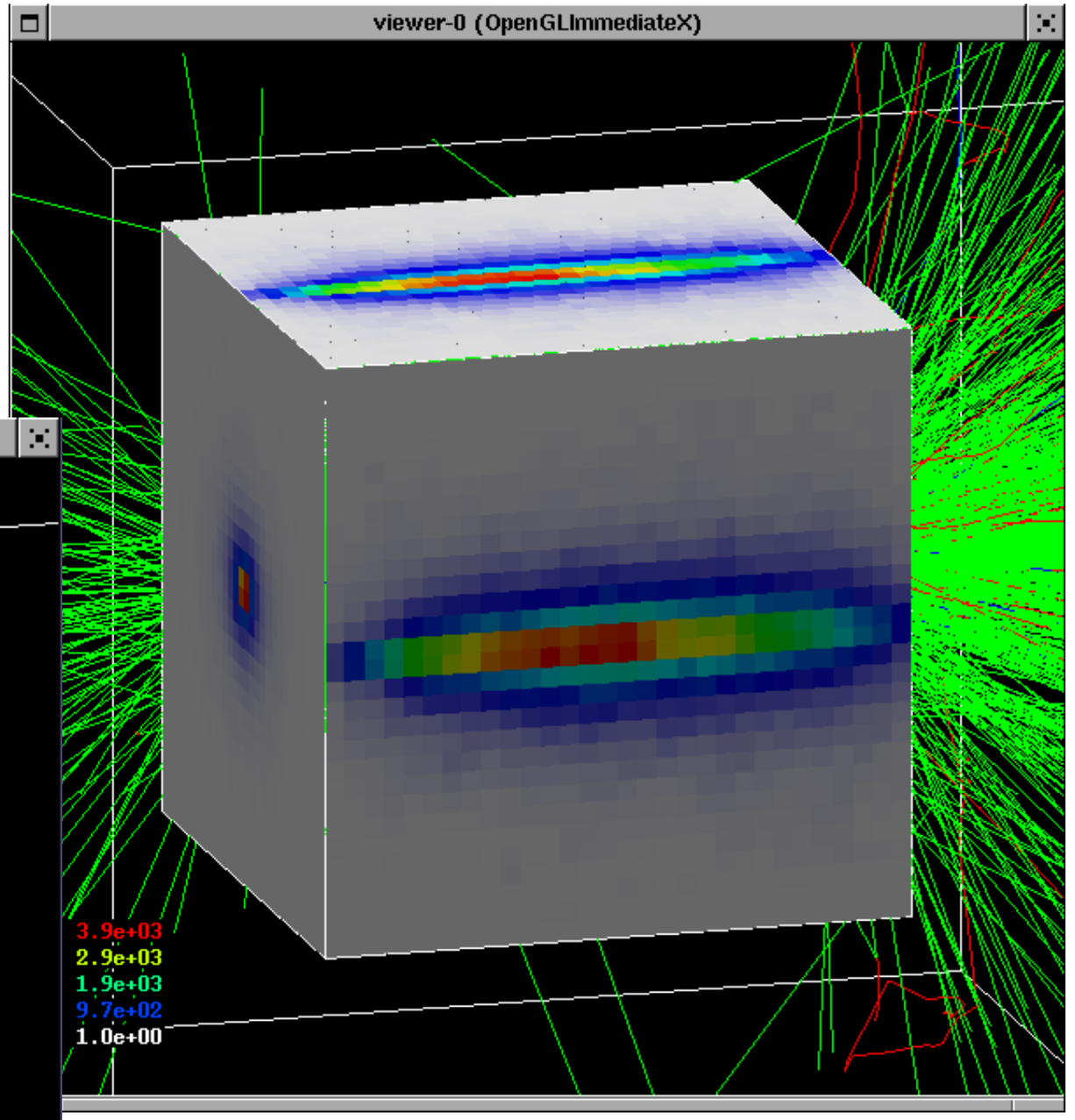
- Define a scoring mesh
  - /score/create/boxMesh <mesh_name>
  - /score/open, /score/close
- Define mesh parameters
  - /score/mesh/boxsize <dx> <dy> <dz>
  - /score/mesh/nbin <nx> <ny> <nz>
  - /score/mesh/translate,
- Define primitive scorers
  - /score/quantity/eDep <scorer_name>
  - /score/quantity/cellFlux <scorer_name>
  - currently **20 scorers** are available
- Define filters
  - /score/filter/particle <filter_name> <particle_list>
  - /score/filter/kinE <filter_name> <Emin> <Emax> <unit>
  - currently **5 filters** are available
- Output
  - /score/draw <mesh_name> <scorer_name>
  - /score/dump, /score/list


viewer-0 (OpenGLImmediateX)

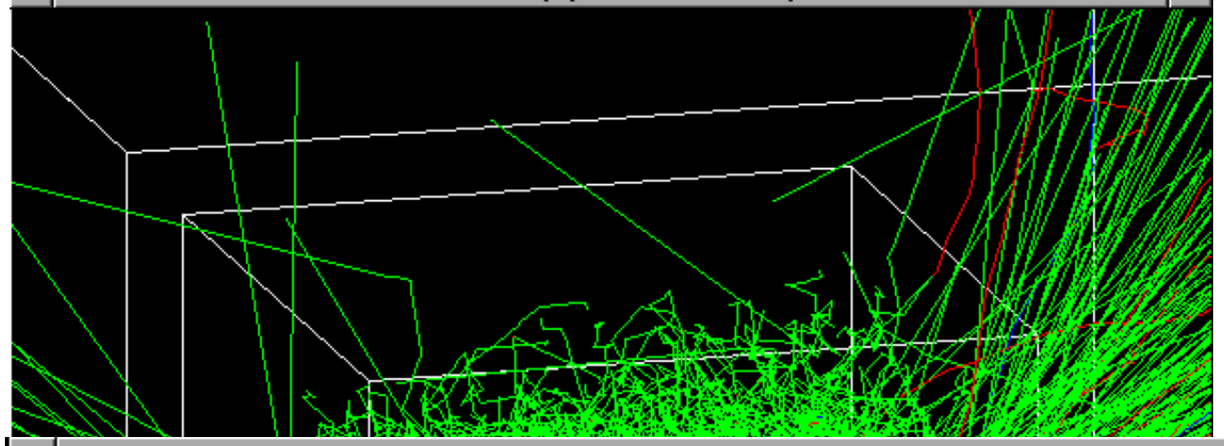

viewer-0 (OpenGLImmediateX)


viewer-0 (OpenGLImmediateX)
3.9e+03
2.9e+03
1.9e+03
9.7e+02
1.0e+00

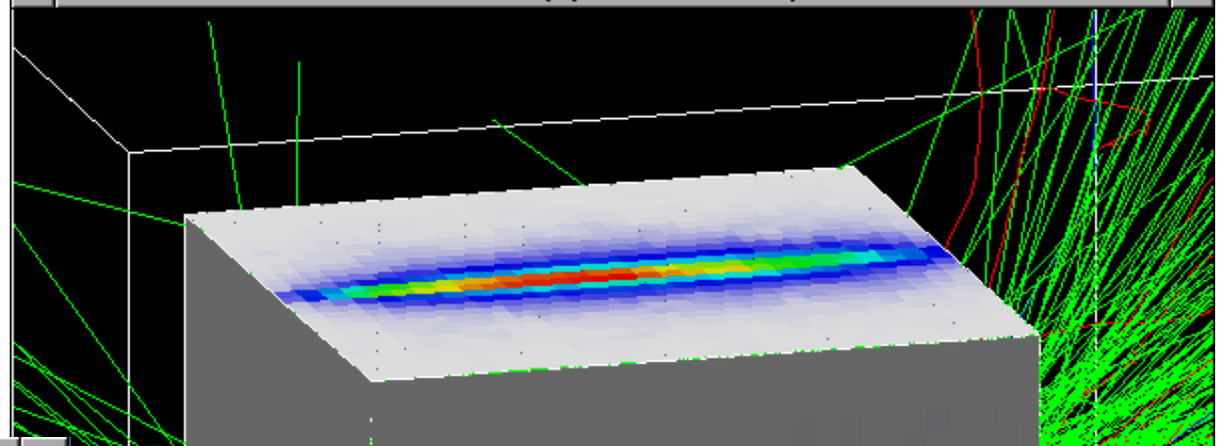

viewer-0 (OpenGLImmediateX)
8.0e+02
6.0e+02
4.0e+02
2.0e+02
0.0e+00


viewer-0 (OpenGLImmediateX)
3.9e+03
2.9e+03
1.9e+03
9.7e+02
1.0e+00

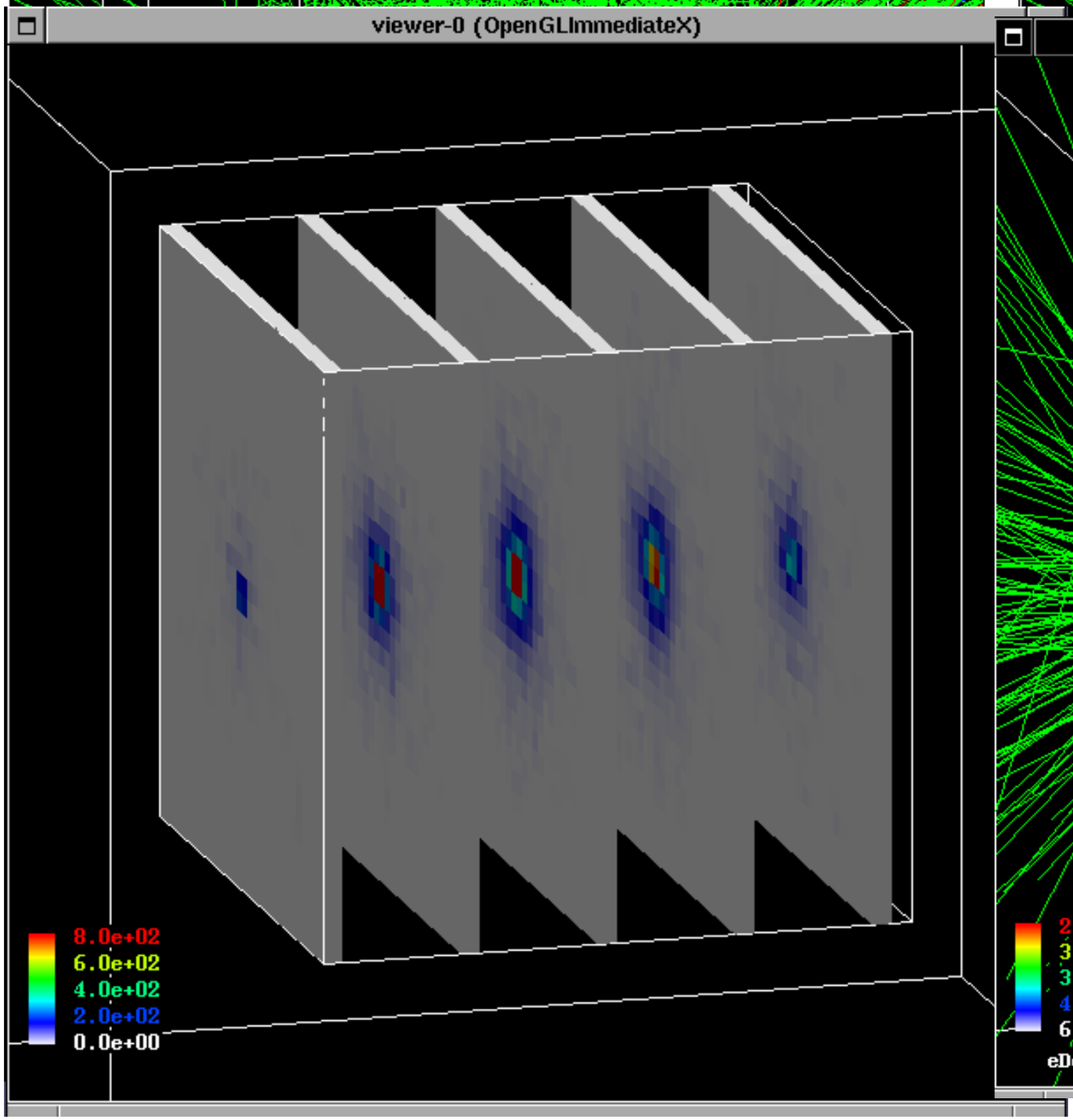
viewer-0 (OpenGLImmediateX)



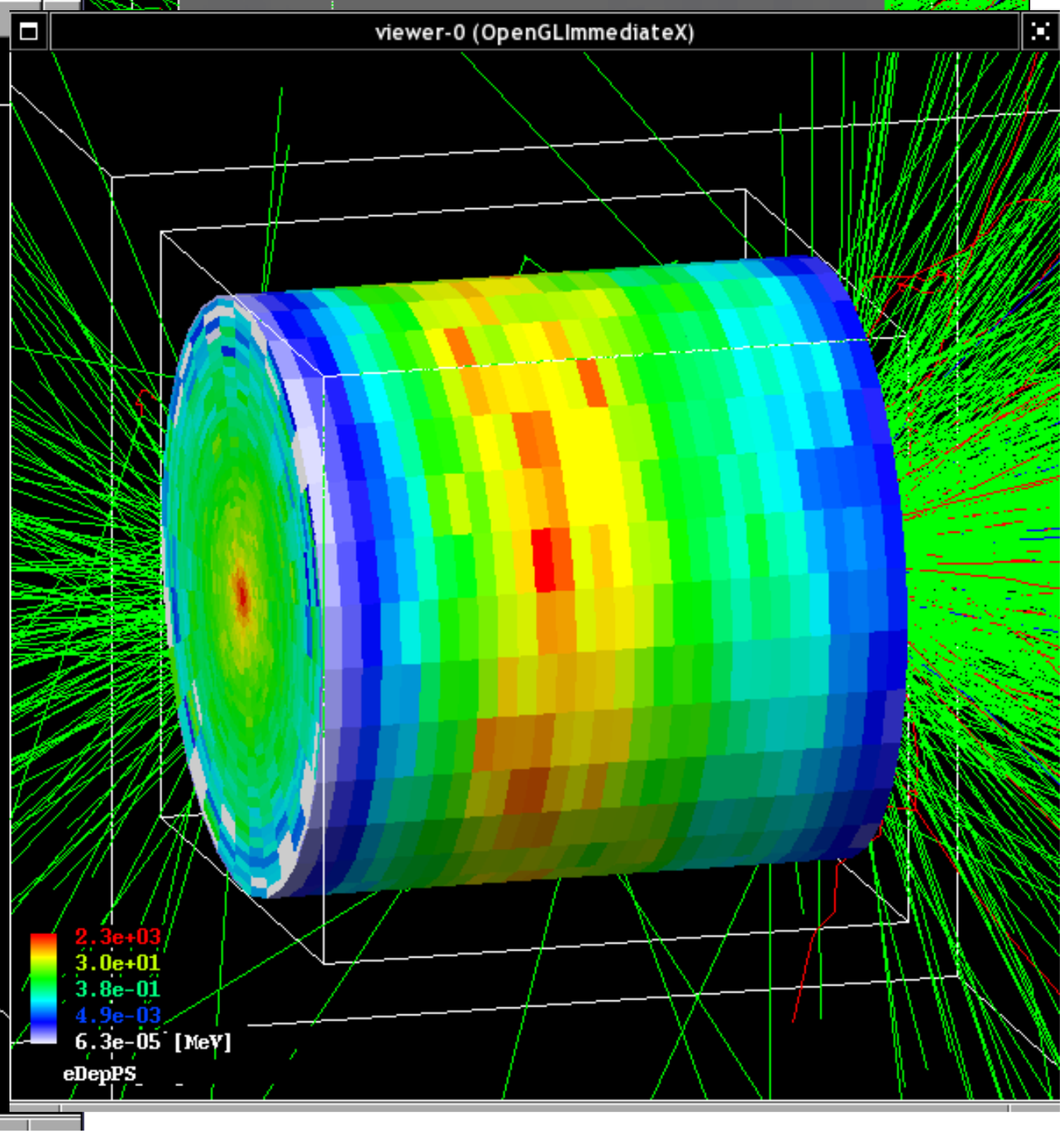
viewer-0 (OpenGLImmediateX)



viewer-0 (OpenGLImmediateX)



viewer-0 (OpenGLImmediateX)



How to learn more

Have a look at the **dedicated extended examples** released with Geant4:

examples/extended/runAndEvent/RE02
(use of primitive scorers)

examples/extended/runAndEvent/RE03
(use of UI-based scoring)