Scoring

Tsukasa Aso
Toyama National College of Technology

Overview

- Overview of scoring in Geant4
- How to use Command-based scoring
- How to define scorers in your geometry
  - How to attach scorers to your geometry
  - How to accumulate scoring results
- Brief introduction for creating your SensitiveDetector (and Hit)
  - How to define your scores/filters
  - How to define your SensitiveDetector and Hit clases

Overview of scoring in Geant4

Extract information

- Geant4 simulates interactions by tracking a particle in matter.
- You need to pick the information up from Geant4.
- There are three ways:
  - Built-in scoring commands
    - Most commonly-used physics quantities are available.
  - Use scorers in your tacking geometry
    - Create scores for each event if you need
    - Create own Run class to record or accumulate scores
  - Develop your G4VSensitiveDetector to a volume to generate “hit”.
    - Use user hooks (G4UserEventAction, G4UserRunAction) to get event / run summary
- You may also use user hooks classes,
  - G4UserTrackingAction, G4UserSteppingAction, etc.
  - You have full access to almost all information
  - Straight-forward, but do-it-yourself

How to use command-line based scoring

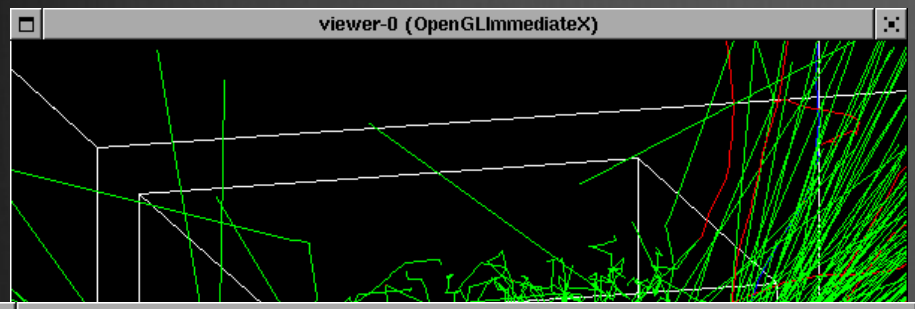
Command-based scoring

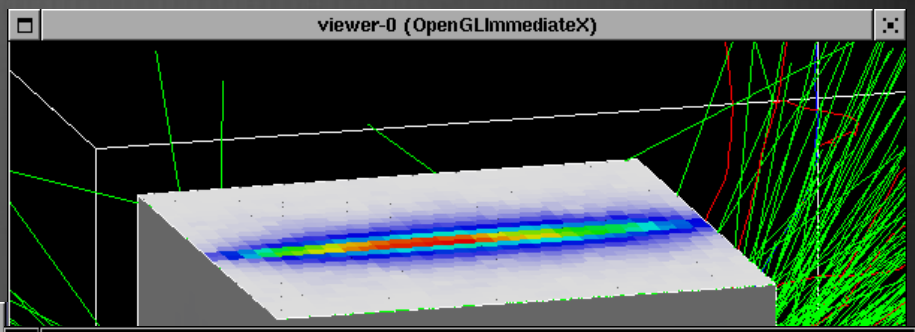
- Command-based scoring functionality offers the built-in scoring mesh and various scorers for commonly-used physics quantities such as dose, flux, etc.
  - Due to small performance overhead, it does not come by default.
- To use this functionality, access to the G4ScoringManager pointer after the instantiation of G4RunManager in your *main*().

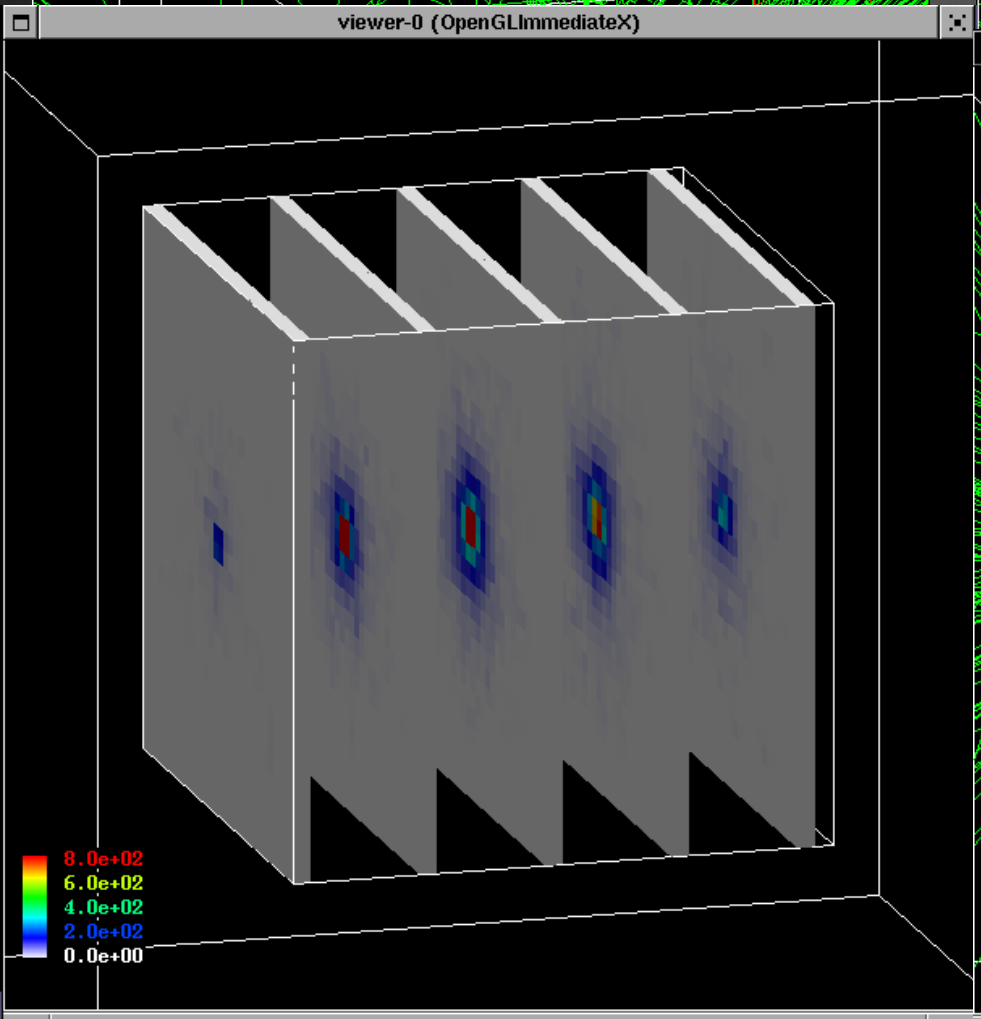
```
#include "G4ScoringManager.hh"
int main()
{
  G4RunManager* runManager = new G4RunManager;
  G4ScoringManager* scoringManager =
                                  G4ScoringManager::GetScoringManager();
  ...
```

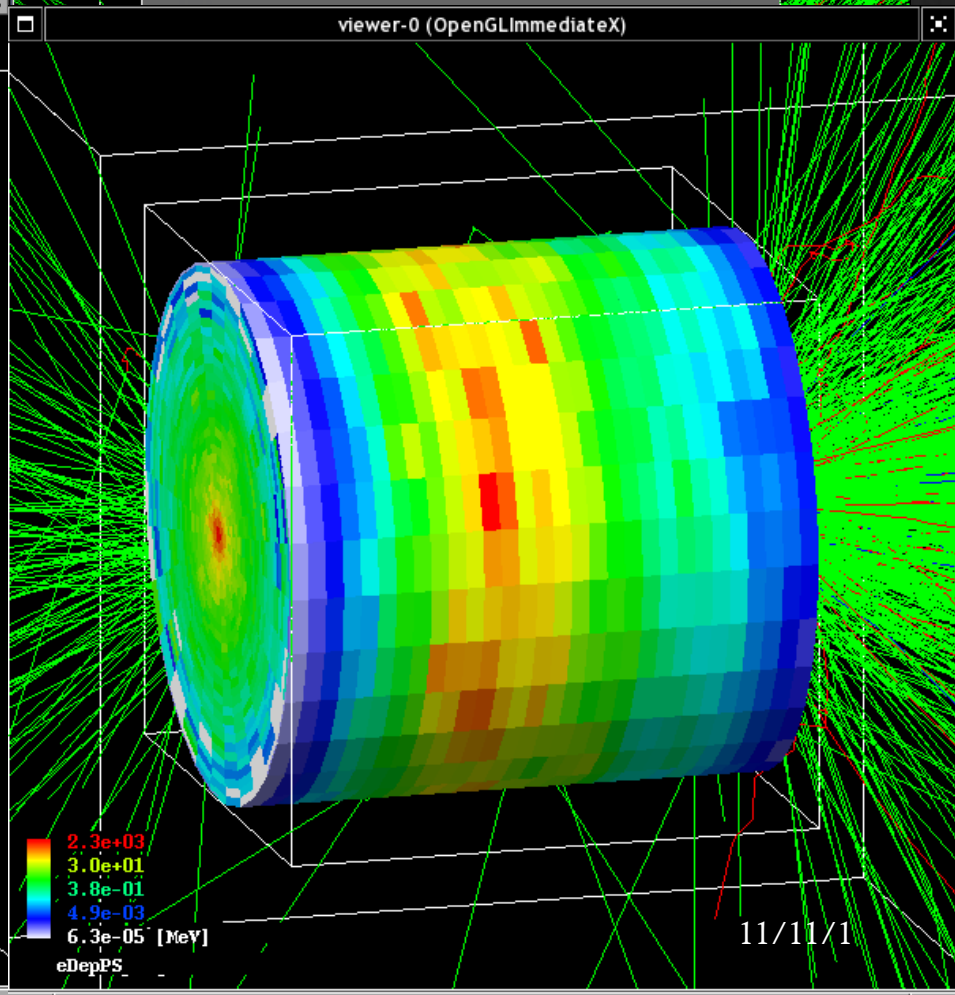
- All of the UI commands of this functionality are in /score/ directory.
- /examples/extended/runAndEvent/RE03

Command-based scorers









Define a scoring mesh

- To define a scoring mesh, the user has to specify the followings.
  1. Shape and name of the 3D scoring mesh.
     - Currently, box and cylinder are available.
  2. Size of the scoring mesh.
     - Mesh size must be specified as "half width" similar to the arguments of G4Box / G4Tubs.
  3. Number of bins for each axes.
     - Note that too many bins causes immense memory consumption.

```
# define scoring mesh
/score/create/boxMesh boxMesh_1
/score/mesh/boxSize 100. 100. 100. cm
/score/mesh/nBin 30 30 30
```

  4. Optionally, position and rotation of the mesh.
     - If not specified, the mesh is positioned at the center of the world volume without rotation.

```
# Translation / Rotation of scoring mesh
/score/mesh/translate    0.   0.   10. cm
/score/mesh/rotate/rotateZ     45.   deg
```

- The mesh geometry can be completely independent to the real material geometry.

Scoring quantities

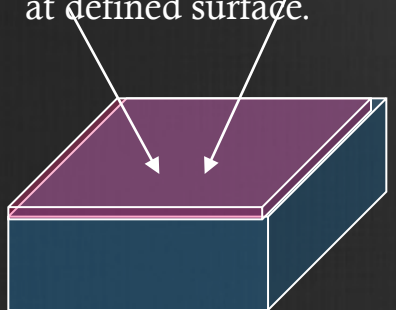
/score/quantitly/xxxxx <scorer_name> <unit>

- A mesh may have arbitrary number of scorers. Each scorer scores one physics quantity.
  - energyDeposit * Energy deposit scorer.
  - cellCharge * Cell charge scorer.
  - cellFlux * Cell flux scorer.
  - passageCellFlux * Passage cell flux scorer
  - doseDeposit * Dose deposit scorer.
  - nOfStep * Number of step scorer.
  - nOfSecondary * Number of secondary scorer.
  - trackLength * Track length scorer.
  - passageCellCurrent * Passage cell current scorer.
  - passageTrackLength * Passage track length scorer.
  - flatSurfaceCurrent * Flat surface current Scorer.
  - flatSurfaceFlux * Flat surface flux scorer.
  - nOfCollision * Number of collision scorer.
  - population * Population scorer.
  - nOfTrack * Number of track scorer.
  - nOfTerminatedTrack * Number of terminated tracks scorer.

List of provided primitive scorers

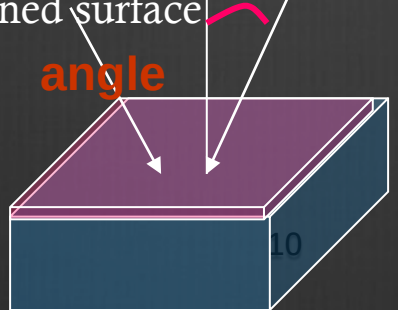
- Concrete Primitive Scorers (See Application Developers Guide 4.4.6)
  - Track length
    - G4PSTrackLength, G4PSPassageTrackLength
  - Deposited energy
    - G4PSEnergyDepsit, G4PSDoseDeposit, G4PSChargeDeposit
  - Current/Flux
    - G4PSFlatSurfaceCurrent, G4PSSphereSurfaceCurrent,G4PSPassageCurrent, G4PSFlatSurfaceFlux, G4PSCellFlux, G4PSPassageCellFlux
  - Others
    - G4PSMinKinEAtGeneration, G4PSNofSecondary, G4PSNofStep

SurfaceCurrent :
Count number of injecting particles at defined surface.

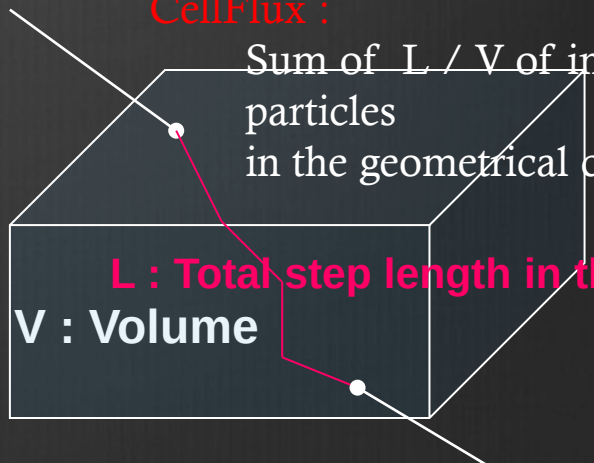


SurfaceFlux :
Sum up 1/cos(angle) of injecting particles at defined surface

angle



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



Command Syntax

- /score/quantity/energyDeposit <scorer name> <unit=MeV>
- /score/quantity/cellCharge <scorer name> <unit=e+>
- /score/quantiy/cellFlux <scorer name> <unit=/cm2>
- /score/quantiy/passageCellFlux <scorer name> <unit=/cm2>
- /score/quantiy/doseDeposit <scorer name> <unit=Gy>
- /score/quantiy/nOfStep <scorer name>
- /score/quantiy/nOfSecondary <scorer name>
- /score/quantiy/trackLength <scorer name> <wflag=F> <kflag=F> <vflag=F> <unit=mm>
  wflag: Multiply track weight , kflag: multiply kinetic energy, vflag: divide by velocity
- /score/quantiy/passageCellCurrent <scorer name> <wflag=T>
- /score/quantiy/passageTrackLength <scorer name> <wflag=T> <unit=mm>
- /score/quantiy/flatSurfaceCurrent <scorer name> <dflag=0> <wflag=T> <aflag=T> <unit=/cm2>
  dflag: 0 InOut, 1 In, 2 Out. aflag : divide by surface true or false
- /score/quantiy/flatSurfaceFlux <scorer name> <dflag=0> <unit=/cm2>
- /score/quantiy/nOfCollision <scorer name> <wflag=F>
- /score/quantiy/population <scorer name> <wflag=F>
- /score/quantiy/nOfTrack <scorer name> <dflag=0> <wflag=F>
- /score/quantiy/nOfTerminatedTrack <scorer name> <wflag=F>

Filter

Filter may be applied to select tracks which have specific conditions

- Each scorer may take a filter. (One scorer can take only one filter.)
  - charged * Charged particle filter.
  - neutral * Neutral particle filter.
  - kineticEnergy * Kinetic energy filter.

    /score/filter/kineticEnergy <fname> <eLow> <eHigh> <unit>
  - particle * Particle filter.

    /score/filter/particle <fname> <p1> … <pn>
  - particleWithKineticEnergy * Particle with kinetic energy filter.

```
/score/quantity/energyDeposit  eDep  MeV
/score/quantity/nOfStep  nOfStepGamma
/score/filter/particle  gammaFilter  gamma
/score/quantity/nOfStep  nOfStepEMinus
/score/filter/particle  eMinusFilter  e-
/score/quantity/nOfStep  nOfStepEPlus
/score/filter/particle  ePlusFilter  e+
/score/close
```

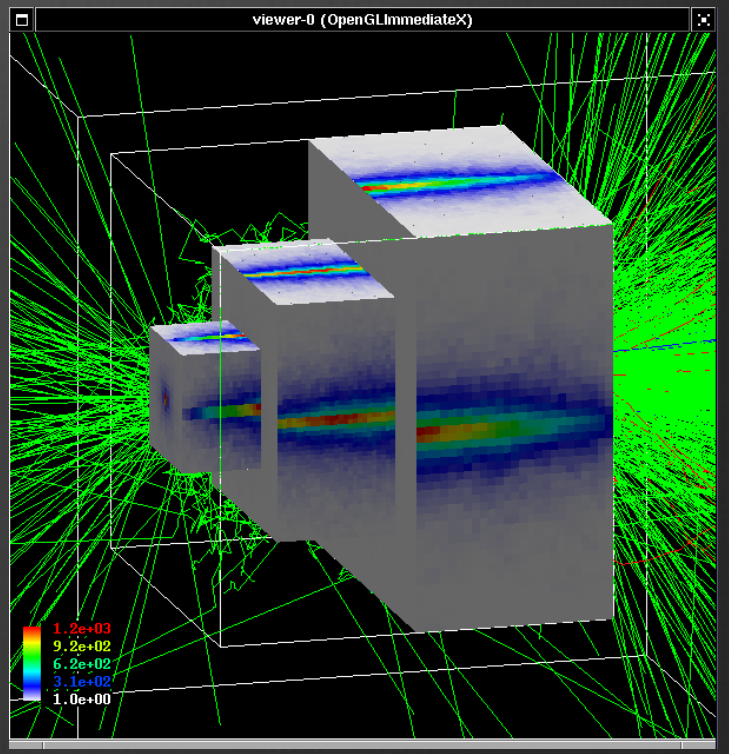
Close the mesh when defining scorers is done.

Example

- $G4INSTALL/examples/extended/runAndEvent/RE03/run2.mac

```
#----------
/score/create/boxMesh boxMesh_2a
/score/mesh/boxSize 30. 30. 20. cm
/score/mesh/translate/xyz 0. 0. -80. cm
/score/mesh/nBin 20 20 20
/score/quantity/nOfStep nOfStepGamma
/score/filter/particle gammaFilter gamma
/score/close
# ----------
/score/create/boxMesh boxMesh_2b
/score/mesh/boxSize 60. 60. 30. cm
/score/mesh/translate/xyz 0. 0. -30. cm
/score/mesh/nBin 50 50 25
/score/quantity/nOfStep nOfStepGamma
/score/filter/particle gammaFilter gamma
/score/close
# ------------
/score/create/boxMesh boxMesh_2c
.....snipped
```

Defining two boxMesh "boxMesh_2a" and "boxMesh_2b" with different size, **location**, and segmentation. The location is given in the center position of ScoringMesh.



Drawing a score

- Projection

/score/drawProjection <mesh_name> <scorer_name> <color_map> <proj=111>

proj: Projection axis, 100 : xy-plane, 010 : yz-place, 001 : zx-plane

- Slice

/score/drawColumn <mesh_name> <scorer_name> <plane> <column> <color_map>

plane: 0 : xy, 1: yz, 2: zx,

- Color map
  - By default, linear and log-scale color maps are available

    "defaultLinearColorMap" or "logColorMap"
  - Minimum and maximum values can be defined by

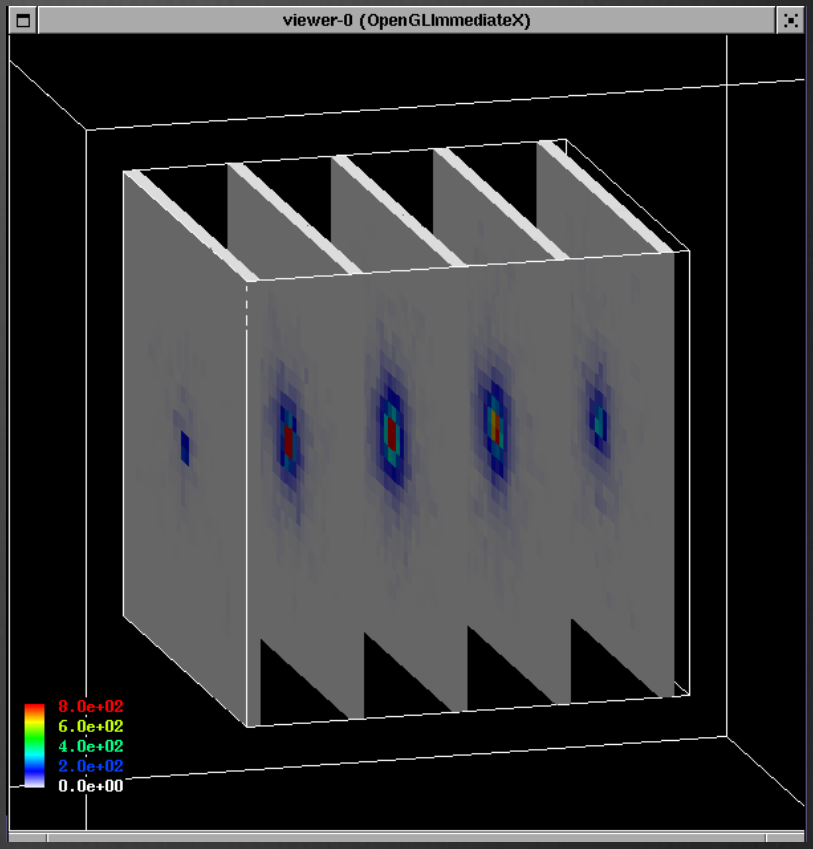
    /score/colorMap/setMinMax <color_map> <min> <max>

    Otherwise, min and max values are taken from the current score.

Example of visualization in command-based scores

- $G4INSTALL/examples/extended/runAndEvent/RE03/run1.mac

```
# drawing projections
/score/drawProjection boxMesh_1 eDep
/score/drawProjection boxMesh_1 nOfStepGamma
/score/drawProjection boxMesh_1 nOfStepEMinus
/score/drawProjection boxMesh_1 nOfStepEPlus
###############
# drawing slices
/vis/scene/create
/vis/sceneHandler/attach scene-2
/score/colorMap/setMinMax ! 0. 800.
/control/loop drawSlice.mac iColumn 0 29 7
```



iColumn = 0, 7, 14, 21, 29

drawSlice.mac

```
/score/drawColumn  boxMesh_1  nOfStepGamma   0   {iColumn}
```

/score/drawColumn <mesh_name> <scorer_name> <plane> <column>

Write scores to a file

- Single score

```
/score/dumpQuantityToFile <mesh_name> <scorer_name> <file_name>
```

- All scores

```
/score/dumpAllQuantitiesToFile <mesh_name> <file_name>
```

- By default, values are written in CSV.
- By creating a concrete class derived from G4VScoreWriter base class, the user can define his own file format.
  - Example in /examples/extended/runAndEvent/RE03
  - User's score writer class should be registered to G4ScoringManager.

Example of file output in command-based scores

- $G4INSTALL/examples/extended/runAndEvent/RE03/

```
# Dump scores to a file
/score/dumpQuantityToFile  boxMesh_1 nOfStepGamma   nOfStepGamma.txt
/score/dumpQuantityToFile  boxMesh_1 eDep           eDep.txt
```

nOfStepGamma.txt

ターミナル —

```
# mesh name: boxMesh_1
# primitive scorer name: nOfStepGamma
0,0,0,0
0,0,1,0
0,0,2,0
0,0,3,7
0,0,4,3
0,0,5,4
0,0,6,1
0,0,7,4
0,0,8,4
0,0,9,1
0,1,0,0
0,1,1,0
```

System of units in G4

millimeter	(mm)
nanosecond	(ns)
Mega electron Volt	(MeV)
positron charge	(eplus)

eDep.txt

ターミナル — more — 40×23

```
:# mesh name: boxMesh_1
# primitive scorer name: eDepPS
# iX, iY, iZ, value [MeV]
0,0,0,0
0,0,1,0
0,0,2,0.01885236634467677
0,0,3,0
0,0,4,0
0,0,5,0
0,0,6,0.002247674811261126
0,0,7,0
0,0,8,0.02565093154574768
0,0,9,0
0,1,0,0
0,1,1,0.007781724926037875
0,1,2,0
0,1,3,0.05388164484436674
0,1,4,0
```

How to define scorers to your geometry

Advantages from command-line scorers

- If you have to be serious about geometry boundary,
  Scorers can store information exactly in the scoring volume.

- If you want to use your own scorers,
  you can develop your scorers and attach it to the scoring volume.

Overview
If you want attach scorers to logical volume in Mass world

- Edit UserDetectorConstruction
  - Attach G4MultiFunctionalDetector to the logical volume
  - Register Primitive Scorers to the G4MultiFunctionalDetctor
  - Occasionally, attach a SDFilter to a Primitve Scorer
  - Here, on your needs, you may define your own scorers or filters in advance
- Edit UserRun
  - Accumulates physical quantities of each event for a run.
- Edit UserRunAction
  - Generate UserRun
  - Dump accumulated quantity

How to attach scorers to your geometry

Example: UserDetectorConstrcution

```
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", fCurrent_In, "percm2");
    myScorer->Register(totalSurfFlux);
    G4VPrimitiveSensitivity* totalDose =
             new G4PSDoseDeposit("TotalDose");
    myScorer->Register(totalDose);

}
```

You may register arbitrary number of primitive scorers.

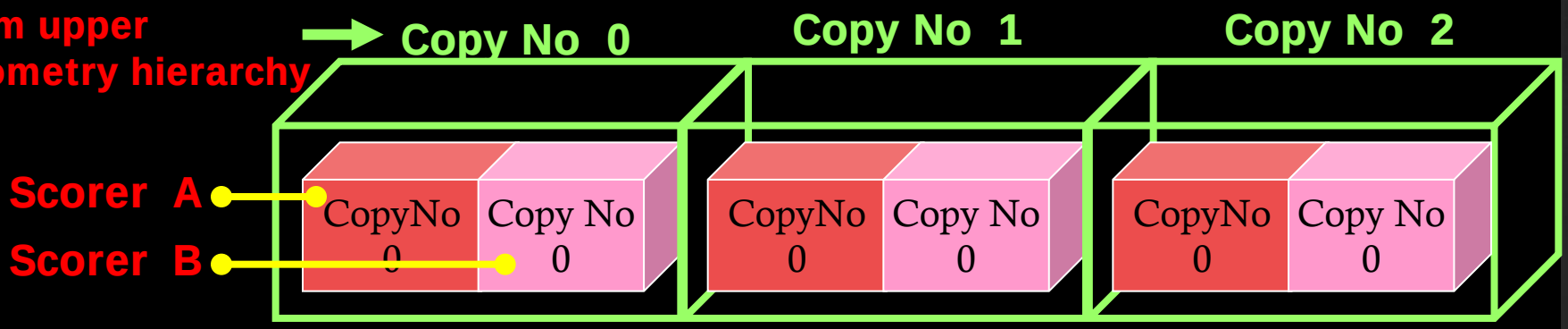
Keys of G4THitsMap

- All provided primitive scorer classes use **G4THitsMap<G4double>**.
- By default, the copy number is taken from the physical volume to which G4MultiFunctionalDetector is assigned.
  - If the physical volume is placed only once, but its (grand-)mother volume is replicated, use the second argument of the constructor of the primitive scorer to indicate the level where the copy number should be taken.

e.g. G4PSCellFlux(G4Steing name, G4String& unit, G4int depth=0)

→ See exampleN07

Key should be taken from upper geometry hierarchy



- If your indexing scheme is more complicated (e.g. utilizing copy numbers of more than one hierarchies), you can override the virtual method GetIndex() provided for all the primitive scorers. For example in 3D hierarchies,
  e.g. **G4PSCellFlux3D::G4PSCellFlux3D(G4String name,const G4String& unit, G4int ni, G4int nj, G4int nk,G4int depi, G4int depj, G4int depk)**

How to accumulate/save results

Score == G4THitsMap<G4double>

- At the end of successful event, G4Event has a vector of G4THitsMap as the scores.
- 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***.

Customized run class(.hh)

```
#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;
public:
    … access methods …
    void  DumpData();    // For example, saving results to file may be here.
};
```

Implement how you accumulate event data

Customized run class (.cc)

```
MyRun::MyRun() : nEvent(0)
{
  G4SDManager* SDM = G4SDManager::GetSDMpointer();
  totalSurfFluxID = SDM->GetCollectionID("myCellScorer/TotalSurfFlux");
  protonSurfFluxID = SDM->GetCollectionID("myCellScorer/ProtonSurfFlux");
  totalDoseID = SDM->GetCollectionID("myCellScorer/TotalDose");
}
```

name of G4MultiFunctionalDetector object

name of G4VPrimitiveSensitivity object

Customized run class (.cc)

```
void MyRun::RecordEvent(const G4Event* evt)
{
  nEvent++;
  G4HCofThisEvent* HCE = evt->GetHCofThisEvent();
  G4THitsMap<G4double>* eventTotalSurfFlux
                        = (G4THitsMap<G4double>*)(HCE->GetHC(totalSurfFluxID));
  G4THitsMap<G4double>* eventProtonSurfFlux
                        = (G4THitsMap<G4double>*)(HCE->GetHC(protonSurfFluxID));
  G4THitsMap<G4double>* eventTotalDose
                        = (G4THitsMap<G4double>*)(HCE->GetHC(totalDoseID));
  totalSurfFlux += *eventTotalSurfFlux;
  protonSurfFlux += *eventProtonSurfFlux;
  totalDose += *eventTotalDose;
}
```

Hit Collection of This Event (HCE)

Get HitsMap by ID

Accumulation (Sum up)

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 …
      theRun->DumpData();   // For example, user defined method for saving results.
}
```

- 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.

Refer to exampleN07

Brief introduction for creating your SensitiveDetector (and Hit)

1) How to create PrimitiveScorers and Filters
2) Overview for developing SensitiveDetector and Hit class

How to
Create your scorer/filter

Creating your own scorer

- Though we provide most commonly-used scorers, you may want to create your own.
  - If you believe your requirement is quite common, just let us know, so that we will add a new scorer.
- G4VPrimitiveScorer is the abstract base class.

```
class G4VPrimitiveScorer
{
 public:
   G4VPrimitiveScorer(G4String name, G4int depth=0);
   virtual ~G4VPrimitiveScorer();
 protected:
   virtual G4bool ProcessHits(G4Step*,
                              G4TouchableHistory*) = 0;
   virtual G4int GetIndex(G4Step*);
 public:
   virtual void Initialize(G4HCofThisEvent*);
   virtual void EndOfEvent(G4HCofThisEvent*);
   virtual void clear();
      …
};
```

Example: G4PSEnergyDeposit

```
G4PSEnergyDeposit::G4PSEnergyDeposit(G4String name, const G4String& unit,
                                     G4int depth)
  :G4VPrimitiveScorer(name,depth),HCID(-1)
{
  SetUnit(unit);
}

G4PSEnergyDeposit::~G4PSEnergyDeposit()
{;}

G4bool G4PSEnergyDeposit::ProcessHits(G4Step* aStep,G4TouchableHistory*)
{
  G4double edep = aStep->GetTotalEnergyDeposit();
  if ( edep == 0. ) return FALSE;
  edep *= aStep->GetPreStepPoint()->GetWeight(); // (Particle Weight)
  G4int  index = GetIndex(aStep);
  EvtMap->add(index,edep);
  return TRUE;
}

void G4PSEnergyDeposit::Initialize(G4HCofThisEvent* HCE)
{
  EvtMap = new G4THitsMap<G4double>(GetMultiFunctionalDetector()->GetName(),
                                    GetName());
  if(HCID < 0) {HCID = GetCollectionID(0);}
  HCE->AddHitsCollection(HCID, (G4VHitsCollection*)EvtMap);
}

void G4PSEnergyDeposit::EndOfEvent(G4HCofThisEvent*)
{;}
```

Unit of quantity should be specified.

G4THitsMap has add() and set() methods

GetIndex( )

G4VPrimitiveScorer (Base class of scorers)

```
G4int G4VPrimitiveScorer::GetIndex(G4Step* aStep)
{
  G4StepPoint* preStep = aStep->GetPreStepPoint();
  G4TouchableHistory* th = (G4TouchableHistory*)(preStep->GetTouchable());
  return th->GetReplicaNumber(indexDepth);
}
```

G4PSEnergyDeposit3D (3D Geometric hierarchy)

```
G4int G4PSEnergyDeposit3D::GetIndex(G4Step* aStep)
{
  const G4VTouchable* touchable = aStep->GetPreStepPoint()->GetTouchable();
  G4int i = touchable->GetReplicaNumber(fDepthi);
  G4int j = touchable->GetReplicaNumber(fDepthj);
  G4int k = touchable->GetReplicaNumber(fDepthk);

  return i*fNj*fNk+j*fNk+k;
}
```

Geometric hierarchy depends on your geometry definition.

Filter class

- G4VSDFilter
  - Abstract base class which you can use to make your own filter

```
class G4VSDFilter
{
  public:
      G4VSDFilter(G4String name);
      virtual ~G4VSDFilter();
  public:
      virtual G4bool Accept(const G4Step*) const = 0;
…
```

Example: G4SDKineticEnergyFilter

```
G4SDKineticEnergyFilter::G4SDKineticEnergyFilter(G4String name,
                                                 G4double elow,
                                                 G4double ehigh)
  :G4VSDFilter(name),fLowEnergy(elow),fHighEnergy(ehigh)
{;}

G4SDKineticEnergyFilter::~G4SDKineticEnergyFilter()
{;}

G4bool G4SDKineticEnergyFilter::Accept(const G4Step* aStep) const
{
  G4double kinetic = aStep->GetPreStepPoint()->GetKineticEnergy();
  if ( kinetic < fLowEnergy  ) return FALSE;
  if ( kinetic >= fHighEnergy ) return FALSE;
  return TRUE;
}
```

If the tack satisfy your condition, then let Accept() return TRUE.

Overview for developing SensitiveDetector and hit

Brief Introduction

- What is the difference between SensitiveDetector/Hit and PrimitiveScorer implementations?
  - It is a design of Hit class.
    - PrimitiveScorer uses a class G4THitsMap<double*>
    - SensitiveDetector/Hit can introduce your own Hit class.
      - The Hit class may have information.
        For example, Tracking Detector following grouped data.
        - Position and time
        - Energy deposition of the step
        - Track ID
      - If you need to analyze correlations of data, it may be useful.

Please look at the detail implementations in
examples/extended/analysis/A01 as an example