WORKSHOP ON MONTE-CAROL SIMULATIONS-APPLICATIONS IN SCIENCE AND TECHNOLOGY
MAY 15-17, 2017
S.AOGAKI

SCORING IN GEANT4
SENSITIVE DETECTOR AND HITS COLLECTION

BEFORE PRESENTATION

- I made one example including my talks
- Please download from
- https://github.com/aogaki/Workshop2017
- e-mail: sohichiroh.aogaki@nipne.ro

CONTENTS

- Introduction
- Sensitive detector and Hits collection
- How to describe
- How to obtain some information

INTRODUCTION

INTRODUCTION

- To obtain some simulation results. We have two ways
  - Using user hooks (G4UserTrackingAction, G4UserSteppingAction, and etc.)
    - You can access full information
    - It is not presented in this presentation
  - Using scoring functionality (G4VSensitiveDetector)
    - I will explain about this
    - My style is very simple

SENSITIVE DETECTOR
AND
HITS COLLECTION

SENSITIVE DETECTOR AND HITS COLLECTION

- Sensitive detector
  - Sensitive detector (G4VSensitiveDetector) is assigned to logical volume (G4LogicalVolume)
  - You can choose any volume
  - Obtaining some informations in ProcessHits (user defining function of sensitive detector)
    - Deposited energy
    - Position information
    - Time information
    - etc.

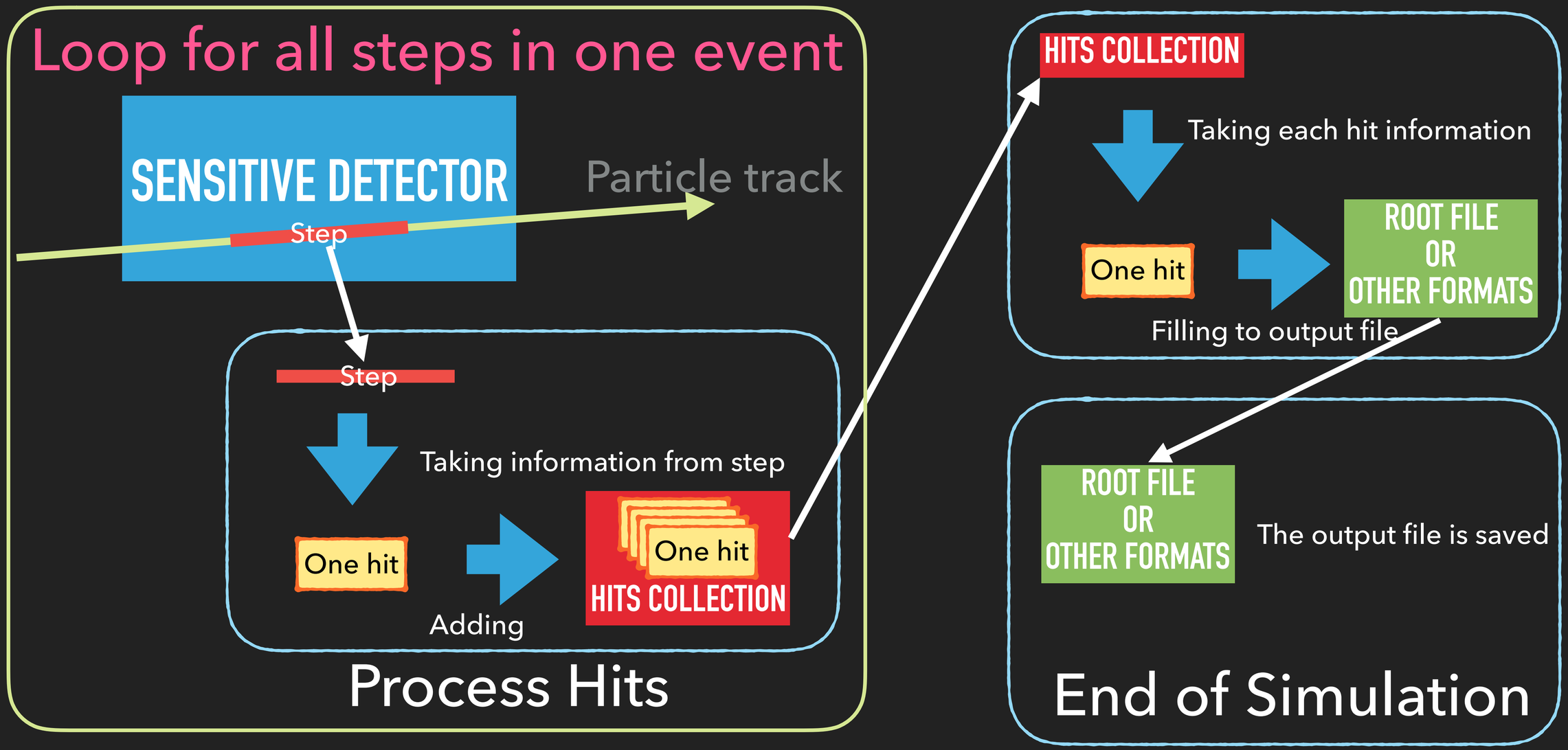
SENSITIVE DETECTOR AND HITS COLLECTION

- Hits collection
  - Like a variable array of hit (G4VHit)
  - One hits collection means one event
  - Each hit has the information come from sensitive detector
  - All information are recorded at the end of event

SENSITIVE DETECTOR AND HITS COLLECTION

- Each logical volume can have the Sensitive detector
  - This logical volume becomes detector!
- Hit is a snapshot of the interaction between particles and the detector
- In one event, there are so many interaction
- Hits collection records all interaction in the detector
  - Sensitive detector make one hit from step (G4Step)

SENSITIVE DETECTOR AND HITS COLLECTION



SENSITIVE DETECTOR AND HITS COLLECTION

- You can make any kind of detector
  - Tracking detector?
    - Recording particle id and position (and momentum)
  - Calorimeter?
    - Recording position (or volume information) and deposited energy

HOW TO DESCRIBE (AND USE IT)

HOW TO DESCRIBE AND USE

- The sensitive detector is assigned for logical volume in DetectorConstruction
- MySD class uses the name of itself “SD” and Hits collection name “HC”.
- Registering detector to G4SDManager
- Assignment sensitive detector to logical volume using the name of logical volume

```
void MyDetectorConstruction::ConstructSDandField()
{
   // Sensitive Detectors
   G4VSensitiveDetector *SD = new MySD("SD", "HC");
   G4SDManager::GetSDMpointer()->AddNewDetector(SD);

   G4LogicalVolumeStore *lvStore = G4LogicalVolumeStore::GetInstance();
   for(auto &&lv: *lvStore){
      if(lv->GetName().contains("LGSORow") ||
         lv->GetName().contains("NaI"))
         SetSensitiveDetector(lv->GetName(), SD);
   }
}
```

HOW TO DESCRIBE AND USE

- SensitiveDetector file
- Initialize()
  - Registering my hits collection with ID number
    - It will be used at the end of an event
- Inside the ProcessHits is next slide

```
MySD::MySD(const G4String &name,
           const G4String &hitsCollectionName)
   : G4VSensitiveDetector(name)
{
   collectionName.insert(hitsCollectionName);
}

void MySD::Initialize(G4HCofThisEvent *hce)
{
   fHitsCollection
      = new MyHitsCollection(SensitiveDetectorName, collectionName[0]);

   G4int hcID
      = G4SDManager::GetSDMpointer()->GetCollectionID(collectionName[0]);
   hce->AddHitsCollection(hcID, fHitsCollection);
}

G4bool MySD::ProcessHits(G4Step *step, G4TouchableHistory */*history*/)
{
  // Obtaining some information
  // and making Hit
}
```

G4CollectionNameVector

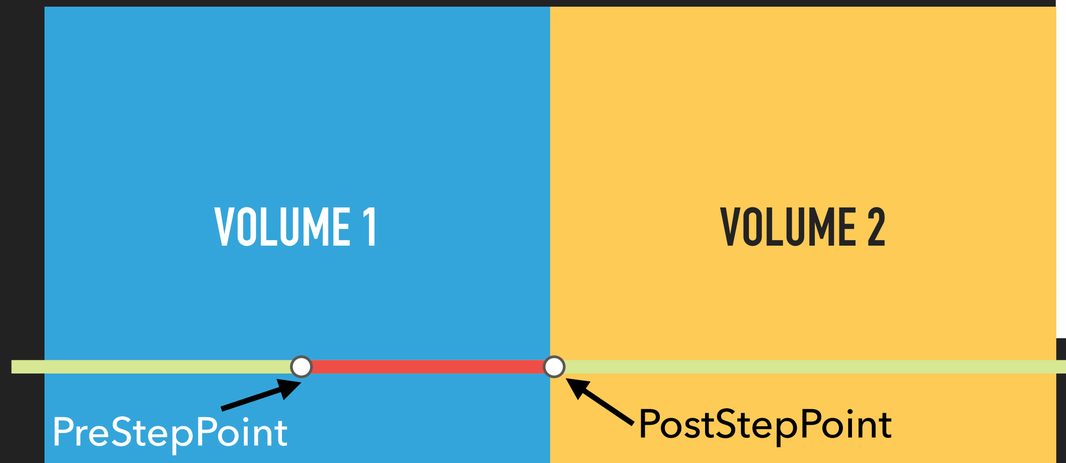
HOW TO DESCRIBE AND USE

- G4Track means all track
- G4StepPoint means end and start points of step
- Usually, we use the information from PostStepPoint
  - We need the information after interaction

```
G4bool MySD::ProcessHits(G4Step *step, G4TouchableHistory */*history*/)
{
   G4Track *track = step->GetTrack();
   G4int trackID = track->GetTrackID();
   //if(trackID != 1) return false; // only the primal particle

   MyHit *newHit = new MyHit();

   newHit->SetTrackID(trackID);

   G4StepPoint *postStepPoint = step->GetPostStepPoint();
   G4ThreeVector position =  postStepPoint->GetPosition();
   newHit->SetPosition(position);

   G4ThreeVector momentum =  postStepPoint->GetMomentum();
   newHit->SetMomentum(momentum);

   G4StepPoint *preStepPoint = step->GetPreStepPoint();
   G4String volumeName = preStepPoint->GetPhysicalVolume()->GetName();
   newHit->SetVolumeName(volumeName);

   fHitsCollection->insert(newHit);
   return true;
}
```

HOW TO DESCRIBE AND USE

- Attention!
- The volume name should be taken from PreStepPoint
- PostStepPoint is at Volume2
- Interaction is in Volume1




```
G4bool MySD::ProcessHits(G4Step *step, G4TouchableHistory */*history*/)
{
   G4Track *track = step->GetTrack();
   G4int trackID = track->GetTrackID();
   //if(trackID != 1) return false; // only the primal particle

   MyHit *newHit = new MyHit();

   newHit->SetTrackID(trackID);

   G4StepPoint *postStepPoint = step->GetPostStepPoint();
   G4ThreeVector position =  postStepPoint->GetPosition();
   newHit->SetPosition(position);

   G4ThreeVector momentum =  postStepPoint->GetMomentum();
   newHit->SetMomentum(momentum);

   G4StepPoint *preStepPoint = step->GetPreStepPoint();
   G4String volumeName = preStepPoint->GetPhysicalVolume()->GetName();
   newHit->SetVolumeName(volumeName);

   fHitsCollection->insert(newHit);
   return true;
}
```

HOW TO DESCRIBE AND USE

```
class MyHit : public G4VHit
{
public:
   MyHit();
   virtual ~MyHit();
   MyHit(const MyHit &right);
   const MyHit &operator=(const MyHit &right);
   int operator==(const MyHit &right) const;

   inline void *operator new(size_t);
   inline void  operator delete(void *);

   // add setter/getter methods
   void SetTrackID(G4int id) {fTrackID = id;};
   G4int GetTrackID() {return fTrackID;};

   void SetPosition(G4ThreeVector pos) {fPosition = pos;};
   G4ThreeVector GetPosition() {return fPosition;};

   void SetMomentum(G4ThreeVector p) {fMomentum = p;};
   G4ThreeVector GetMomentum() {return fMomentum;};

   void SetVolumeName(G4String volumeName) {fVolumeName = volumeName;};
   G4String GetVolumeName() {return fVolumeName;};

private:
   G4int fTrackID;
   G4ThreeVector fPosition;
   G4ThreeVector fMomentum;
   G4String fVolumeName;
};
```

- Hit and Hits collection file
- Hit class has setter and getter methods
- In the same file my Hits collection class is also defined

```
typedef G4THitsCollection<MyHit> MyHitsCollection;

extern G4ThreadLocal G4Allocator<MyHit> *MyHitAllocator;

inline void *MyHit::operator new(size_t)
{
   if (!MyHitAllocator)
      MyHitAllocator = new G4Allocator<MyHit>;
   return (void *)MyHitAllocator->MallocSingle();
}

inline void MyHit::operator delete(void *hit)
{
   MyHitAllocator->FreeSingle((MyHit *) hit);
}

#endif
```

HOW TO DESCRIBE AND USE

- For multi threading, G4Allocator should be G4ThreadLocal
  - Simply say, MyHitAllocator will be made for each thread
- This definition of new and delete operator is a good trick
  - If using new and delete at each hit, the computational cost of it is very big
  - Thus, delete don't delete memory space
  - And, new uses recycled memory space
- Hit and Hits collection file
- Hit class has setter and getter methods
- In the same file my Hits collection class is also defined

```
typedef G4THitsCollection<MyHit> MyHitsCollection;

extern G4ThreadLocal G4Allocator<MyHit> *MyHitAllocator;

inline void *MyHit::operator new(size_t)
{
   if (!MyHitAllocator)
      MyHitAllocator = new G4Allocator<MyHit>;
   return (void *)MyHitAllocator->MallocSingle();
}

inline void MyHit::operator delete(void *hit)
{
   MyHitAllocator->FreeSingle((MyHit *) hit);
}

#endif
```

HOW TO DESCRIBE AND USE

- Event action
- Before the ned of event

```
MyEventAction::MyEventAction()
   : G4UserEventAction(),
     fHitsCollectionID(-1)
{}

MyEventAction::~MyEventAction()
{}

MyHitsCollection *MyEventAction::GetHitsCollection(G4int hcID, const G4Event *event)
const
{
   MyHitsCollection *hitsCollection
      = static_cast<MyHitsCollection *>(
         event->GetHCofThisEvent()->GetHC(hcID));

   if ( ! hitsCollection ) {
     // Some error handling
   }

   return hitsCollection;
}

void MyEventAction::BeginOfEventAction(const G4Event *)
{}
```

HOW TO DESCRIBE AND USE

- EndOfEventAction
- I record information hit by hit (step by step)
- You can do event by event
- For example, summing all deposited energy

```
void MyEventAction::EndOfEventAction(const G4Event *event)
{
   if (fHitsCollectionID == -1)
      fHitsCollectionID = G4SDManager::GetSDMpointer()->GetCollectionID("HC");

   MyHitsCollection *hc = GetHitsCollection(fHitsCollectionID, event);

   G4int eventID = event->GetEventID();

   G4AnalysisManager *anaMan = G4AnalysisManager::Instance();

   const G4int kHit = hc->entries();
   for (G4int iHit = 0; iHit < kHit; iHit++) {
      MyHit *newHit = (*hc)[iHit];

      anaMan->FillNtupleIColumn(0, 0, eventID); // EventID

      G4int trackID = newHit->GetTrackID();
      anaMan->FillNtupleIColumn(0, 1, trackID);

      G4String volumeName = newHit->GetVolumeName();
      anaMan->FillNtupleSColumn(0, 2, volumeName);

      G4ThreeVector position = newHit->GetPosition();
      anaMan->FillNtupleDColumn(0, 3, position.x());
      anaMan->FillNtupleDColumn(0, 4, position.y());
      anaMan->FillNtupleDColumn(0, 5, position.z());

      G4ThreeVector momentum = newHit->GetMomentum();
      anaMan->FillNtupleDColumn(0, 6, momentum.x());
      anaMan->FillNtupleDColumn(0, 7, momentum.y());
      anaMan->FillNtupleDColumn(0, 8, momentum.z());

      anaMan->AddNtupleRow(0);
   }

}
```

HOW TO DESCRIBE AND USE

- Run action
- Simply, Before run, making Ntuple
- After run, file writing

```
#include "g4root.hh"
void MyRunAction::BeginOfRunAction(const G4Run *)
{
   G4AnalysisManager *anaMan = G4AnalysisManager::Instance();
   G4String fileName = "result";
   anaMan->OpenFile(fileName);

   // Interaction
   anaMan->CreateNtuple("hits", "test detector");
   anaMan->CreateNtupleIColumn(0, "EventID");
   anaMan->CreateNtupleIColumn(0, "TrackID");
   anaMan->CreateNtupleSColumn(0, "VolumeName");

   anaMan->CreateNtupleDColumn(0, "x");
   anaMan->CreateNtupleDColumn(0, "y");
   anaMan->CreateNtupleDColumn(0, "z");

   anaMan->CreateNtupleDColumn(0, "vx");
   anaMan->CreateNtupleDColumn(0, "vy");
   anaMan->CreateNtupleDColumn(0, "vz");

   // Init parameters
   anaMan->CreateNtuple("InitPar", "Initial Parameters");
   anaMan->CreateNtupleIColumn(1, "EventID");
   anaMan->CreateNtupleIColumn(1, "PDGCode");
   anaMan->CreateNtupleDColumn(1, "KineticEnergy");
   anaMan->CreateNtupleDColumn(1, "vx");
   anaMan->CreateNtupleDColumn(1, "vy");
   anaMan->CreateNtupleDColumn(1, "vz");

   anaMan->FinishNtuple();
}

void MyRunAction::EndOfRunAction(const G4Run *)
{
   G4AnalysisManager *anaMan = G4AnalysisManager::Instance();
   anaMan->Write();
   anaMan->CloseFile();
}
```

HOW TO OBTAIN SOME INFORMATION (EXAMPLES)

HOW TO OBTAIN SOME INFORMATION (EXAMPLES)

- The list of information
- Particle information
  - Global and local position
  - Track ID and ParentID
  - PDG code and particle name
  - Momentum
  - Kinetic energy
  - Vertex volume name

HOW TO OBTAIN SOME INFORMATION (EXAMPLES)

- The list of information
- Particle information
  - Global and local position
  - Global means position in whole setup
  - Local means position in the logical volume
  - You can obtain its from pre step point (GetPreStepPoint())

```
G4bool MySD::ProcessHits(G4Step *step, G4TouchableHistory */*history*/)
{
   G4StepPoint *postStepPoint = step->GetPostStepPoint();

   G4ThreeVector position =  postStepPoint->GetPosition();
   newHit->SetPosition(position);
   G4TouchableHandle theTouchable = postStepPoint->GetTouchableHandle();
   G4ThreeVector localPosition = theTouchable->GetHistory()->
                                 GetTopTransform().TransformPoint(position);
}
```

HOW TO OBTAIN SOME INFORMATION (EXAMPLES)

- The list of information
- Particle information
  - Track ID
    - 1 is primal
    - More than 2 means daughter particles
  - Parent ID
    - The parent of primal particle is 0

```
G4bool MySD::ProcessHits(G4Step *step, G4TouchableHistory */*history*/)
{
   G4Track *track = step->GetTrack();
   G4int trackID = track->GetTrackID();
   G4int parentID = track->GetParentID();
}
```

HOW TO OBTAIN SOME INFORMATION (EXAMPLES)

- The list of information
- Particle information
  - PDG code and particle name

```
G4bool MySD::ProcessHits(G4Step *step, G4TouchableHistory */*history*/)
{
   G4Track *track = step->GetTrack();
   G4ParticleDefinition *particle = track->GetDefinition();
   G4int pdgCode = particle->GetPDGEncoding();
   G4String parName = particle->GetParticleName();
}
```

HOW TO OBTAIN SOME INFORMATION (EXAMPLES)

- The list of information
- Particle information
  - Momentum
  - Also you can take from pre step point

```
G4bool MySD::ProcessHits(G4Step *step, G4TouchableHistory */*history*/)
{
   G4StepPoint *postStepPoint = step->GetPostStepPoint();
   G4ThreeVector momentum =  postStepPoint->GetMomentum();
}
```

HOW TO OBTAIN SOME INFORMATION (EXAMPLES)

- The list of information
- Particle information
  - Kinetic energy

```
G4bool MySD::ProcessHits(G4Step *step, G4TouchableHistory */*history*/)
{
   G4StepPoint *postStepPoint = step->GetPostStepPoint();
   G4double kineticEnergy = postStepPoint->GetKineticEnergy();
}
```

HOW TO OBTAIN SOME INFORMATION (EXAMPLES)

- The list of information
- Particle information
  - Vertex volume name
  - Sometime, this helps to find the noise source

```
G4bool MySD::ProcessHits(G4Step *step, G4TouchableHistory */*history*/)
{
   G4Track *track = step->GetTrack();
   G4String vertexName = track->GetLogicalVolumeAtVertex()->GetName();
}
```

HOW TO OBTAIN SOME INFORMATION (EXAMPLES)

- The list of information
- Volume information
  - Volume name
  - Step point is boundary or not
  - Deposited energy
  - Copy number of volume

HOW TO OBTAIN SOME INFORMATION (EXAMPLES)

- The list of information
- Volume information
  - Volume name
    - It should be taken from pre step point

```
G4bool MySD::ProcessHits(G4Step *step, G4TouchableHistory */*history*/)
{
   G4StepPoint *preStepPoint = step->GetPreStepPoint();
   G4String volumeName = preStepPoint->GetPhysicalVolume()->GetName();
}
```

HOW TO OBTAIN SOME INFORMATION (EXAMPLES)

- The list of information
- Volume information
  - Step point is boundary or not
  - If pre step point is boundary, It means incident point
  - If post step point is boundary, it means particle is exit from volume

```
G4bool MySD::ProcessHits(G4Step *step, G4TouchableHistory */*history*/)
{
   G4StepPoint *postStepPoint = step->GetPostStepPoint();
   G4int isExit = (postStepPoint->GetStepStatus() == fGeomBoundary);
   if(isExit == 0) return false; // only going out particle
   MyHit *newHit = new MyHit();
   newHit->SetIsExit(isExit);

   G4StepPoint *preStepPoint = step->GetPreStepPoint();
   if(preStepPoint->GetStepStatus() == fGeomBoundary){
      G4double incidentEnergy = preStepPoint->GetKineticEnergy();
      newHit->SetIncidentEnergy(incidentEnergy);
   }
   else newHit->SetIncidentEnergy(0.);
}
```

HOW TO OBTAIN SOME INFORMATION (EXAMPLES)

- The list of information
- Volume information
  - Deposited energy

```
G4bool MySD::ProcessHits(G4Step *step, G4TouchableHistory */*history*/)
{
   G4double depositEnergy = step->GetTotalEnergyDeposit();
   newHit->SetDepositEnergy(depositEnergy);
}
```

HOW TO OBTAIN SOME INFORMATION (EXAMPLES)

- The list of information
- Volume information
  - Copy number
  - It is useful for like a SciFi detector
  - This number represents the which fiber
  - Of course, you have to set the number, when you construct the volume

```
G4bool MySD::ProcessHits(G4Step *step, G4TouchableHistory */*history*/)
{
   if(volumeName == "LGSORow"){
      G4TouchableHandle touch = preStepPoint->GetTouchableHandle();
      G4int copyNo = touch->GetCopyNumber();
      G4int motherCopyNo = touch->GetCopyNumber(1);
      G4cout << motherCopyNo <<"\t"<< copyNo << G4endl;
   }
}
```

CONCLUSION

- Sensitive detector (SD) is assigned in Detector construction class
- SD needs the name of Hits collection (HC), when it is registered
- User defines ProcessHits member function of SD
  - It takes some information as Hit
  - Each Hit is added to HC
- End of each event, HC is read and its information are filled into an output file
- When simulation is finished, all information is written in the output file

APPENDIX: HOW TO ANALYZE BY ROOT

- Using ROOT
- Making a 2D histogram showing the projection of interaction position and deposited energy at NaI detector
- First, making simulation data
- The working directory is Workshop2017
- After compiling example and running example with 1 million events

APPENDIX: HOW TO ANALYZE BY ROOT

- Using chain and make analyzer class
- root [0]: defining TChain and setting ntuple name
- root[1]: Adding ROOT file
- root[2]: Making files

```
HappyNutty:Workshop2017 aogaki$ cd samples/
HappyNutty:samples aogaki$ root
   ------------------------------------------------------------------
  | Welcome to ROOT 6.09/01                        http://root.cern.ch |
  |                                      (c) 1995-2016, The ROOT Team |
  | Built for macosx64                                                |
  | From heads/master@v6-09-01-1515-g201dc83298, Feb 15 2017, 13:52:00 |
  | Try '.help', '.demo', '.license', '.credits', '.quit'/'.q'        |
   ------------------------------------------------------------------

root [0] TChain *chain = new TChain("hits");
root [1] chain->Add("../result_t*.root");
root [2] chain->MakeSelector("MyAnalyzer")
(int) 0
root [3] .q
```

APPENDIX: HOW TO ANALYZE BY ROOT

- MyAnalysis.C and MyAnalysis.h are created by ROOT
- runAnalyzer.cpp runs MyAnalysis

```
HappyNutty:samples aogaki$ ls
MakeROOTfile.cpp  MyAnalyzer.h
MyAnalyzer.C      runAnalyzer.cpp
```

APPENDIX: HOW TO ANALYZE BY ROOT

- runAnalyzer.cpp loads files and run MyAnalysis
- Using TProofLite
- TProofLite is the class of multi-threading

```
#include <TChain.h>
#include <TProof.h>
#include <TProofLite.h>


void ActivatePROOF(TChain *chain, Int_t nThreads = 0)
{
   TProof *proof = TProof::Open("");
   proof->SetProgressDialog(kFALSE);
   if(nThreads > 0) proof->SetParallel(nThreads);

   chain->SetProof();
}

void runAnalyzer()
{
   TChain *chain = new TChain("hits");
   chain->Add("../result_t*.root");
   ActivatePROOF(chain);

   chain->Process("MyAnalyzer.C+O");
}
```

APPENDIX: HOW TO ANALYZE BY ROOT

- Defining a histogram
- It should be nullptr at Begin()
- In SlaveBegin(), it is created
- GetOutputList()->Add(fHisNaI); is needed to save
- In Terminate, the histogram is written and saved

```
class MyAnalyzer : public TSelector {
public :
   …
   TH1D *fHisNaI;
   …
};
```

MyAnalyzer.h

```
void MyAnalyzer::Begin(TTree * /*tree*/)
{
   …
   fHisNaI = nullptr;
}

void MyAnalyzer::SlaveBegin(TTree * /*tree*/)
{
   …
   fHisNaI = new TH1D("HisNaI", "test", 200, -20., 0.,);
   fHisNaI->SetXTitle("z [mm]");
   fHisNaI->SetYTitle("Deposited energy [MeV]");
   GetOutputList()->Add(fHisNaI);
}

void MyAnalyzer::Terminate()
{
   …
   TFile *file = new TFile("output.root", "RECREATE");
   fHisNaI->Write();
   file->Close();
}
```

MyAnalyzer.C

APPENDIX: HOW TO ANALYZE BY ROOT

- Analyzing
- If volume name is NaI
- Recording depth and deposited energy

```
Bool_t MyAnalyzer::Process(Long64_t entry)
{
   …
   TString detName = &(VolumeName[0]);
   if(detName == "NaI"){
     fHisNaI->Fill(*z, *DepositEnergy);
   }
   …
}
```

APPENDIX: HOW TO ANALYZE BY ROOT

- Finally, you obtain the result