GEANT4 Introduction

Blake Leverington

UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

A toolkit

- **Geant4 is a toolkit** for simulating the passage of particles through matter
- each simulation problem requires different configuration (geometry, scoring, particles, physics, etc..) that the user needs to define
- example applications that Geant4 provides are in the /examples folder

The complete beginner Tutorial course can be found here:
https://indico.cern.ch/event/865808/
With more advanced topics here:
https://indico.cern.ch/event/789510/timetable/#20190326

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

The Manual

- https://geant4-userdoc.web.cern.ch/UsersGuides/ForApplicationDeveloper/html/index.html

Physics Processes

https://geant4-userdoc.web.cern.ch/UsersGuides/ForApplicationDeveloper/html/TrackingAndPhysics/physicsProcess.html

Major catagories:

1. electromagnetic,
   - Ionisation and delta ray production
   - Multiple scattering
2. hadronic,
3. decay,
4. photolepton-hadron,
5. Optical
   - Cherenkov
   - Scintillation
   - Rayleigh Scattering
   - Wavelength Shifting
   - Boundary Process
6. parameterization, and
7. transportation

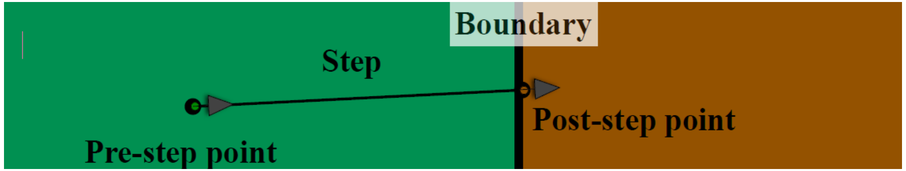
- Many alternative physics models depending on the energy and physics of interest
  - Medical/DNA, micro-electronics, etc
  - Ions
- Based on experimental measurements
- It's important to understand which model is the most applicable to the problem to produce accurate results

KEY CONCEPTS

- a **G4Track** object represents/describes the state of a particle that is under simulation in a given instant of the time (i.e. a given time point)
  - the **G4Track** object is propagated in a step-by-step way during the simulation and the dynamic properties are continuously updated to reflect the current state

- a **G4Step** object can provide the information regarding the change in the state of the particle (that is under tracking) within a simulation step (i.e. delta)

- a **G4Event** is the basic simulation unit that represents a **set of G4Track objects**



Where are we?
Materials and Geometry

What can happen along this step?
Physics Processes

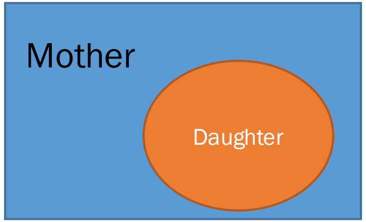
- **G4Event:**
  - a `G4Event` is the basic simulation unit that represents a **set of `G4Track` objects**
  - **at the beginning** of an event:
    - **primary G4Track object**(s) is(are) **generated** (with their static and initial dynamic properties) and pushed **to a track-stack**
    - **one G4Track object** is **popped from** this track-**stack** and transported/tracked/simulated:
      - the track object **is propagated in a step-by-step way** and its dynamic properties as well as the corresponding `G4Step` object are updated at each step
      - the step is limited either by physics interaction or geometry boundary
      - **transportation** (to the next volume through the boundary) will take place in the later while **physics** interaction in the former case
      - **secondary G4Track-s**, **generated by** these **physics interactions**, are **also pushed to** the track-**stack**
      - a **G4Track object** is kept tracking till:
        - + leaves the outermost (World) volume i.e. goes out of the simulation universe
        - + participates in a destructive interaction (e.g. day or photoelectric absorption)
        - + its kinetic energy becomes zero and doesn't have interaction that can happen "at-rest"
        - + the user decided to (artificially) stop the tracking and kill
      - when one track object reaches its termination point, **a new G4Track object** (either secondary or primary) is popped from the stack for tracking
    - processing an event will be terminated when there is **no any G4Track objects in the track-stack**
  - **at the end** of an event, the corresponding `G4Event` object will store its input i.e. the list of primaries (and possible some of its outputs like hits or trajectory collection)

- **G4Run** is a collection of **G4Event-s** (a **G4Event** is a collection of **G4Track**-s)
  - during a run, events are taken and processed one by one in an event-loop
  - **before the start of a run** i.e. at run initialisation (G4RunManager::Initialize()): the **geometry** is **constructed** and **physics** is **initialised**
  - **at the start of a run** (G4RunManager::BeamOn()): the **geometry is optimised** for tracking (voxelization), the event processing starts i.e. entering into the event-loop

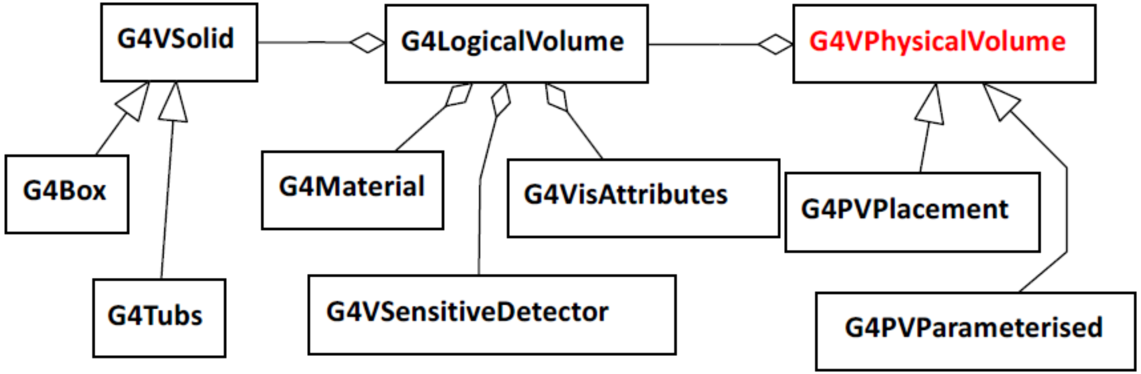
Geometry

Detector geometry

- Three conceptual layers
  - G4VSolid -- *shape, size*
  - G4LogicalVolume -- *daughter physical volumes,*
    material, sensitivity, user limits, etc.
  - G4VPhysicalVolume -- *position, rotation*



"old" terminology for heirarchy



Many more advanced ways of describing and placing geometry

A basic program

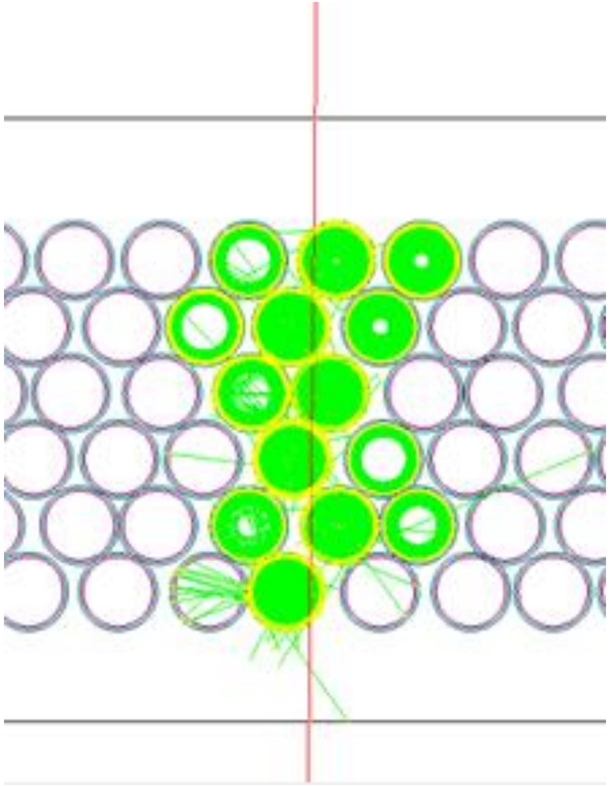
- **user needs to provide these** settings **in the `main()` method** of their **Geant4** application
  - scifiSim.cc in our program
- Create a **G4RunManager** object **(mandatory):**
  - the **only** mandatory **manager object** that user needs to create: all others (**G4EventManager**,**G4SteppingManger**, etc.) are created and deleted automatically
  - all problem specific information need to be given to the **G4RunManager** by the user through the
  - interfaces provided by the **Geant4** toolkit (we will see them one by one):
  - **G4VUserDetectorConstruction**(mandatory): how the geometry should be constructed, built
  - **G4VUserPhyscsList**(mandatory): all the particles and their physics interactions to be simulated
  - **G4VUserActionInitialization** (mandatory):
    - ✴ **G4VUserPrimaryGeneratorAction** (mandatory): how the primary particle(s) in an event should be produced
    - ✴ additional, **optional user actions** (**G4UserRunAction**, **G4UserEventAction**, **G4UserSteppingAction**, etc..)

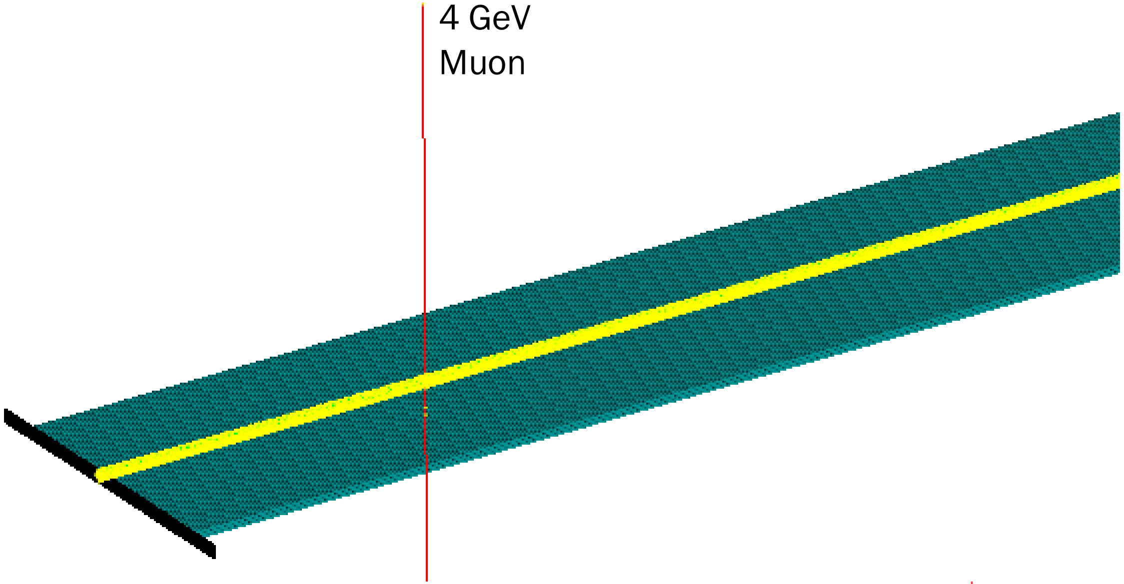
SciFiMatG4_v2

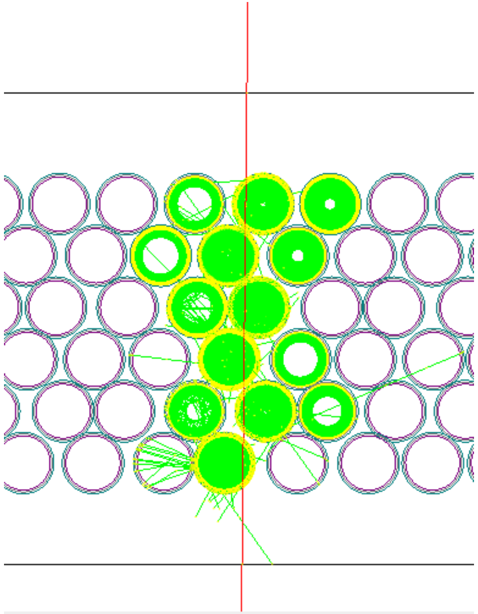
A Geant4 Simulation of a Scintillating Fibre Mat

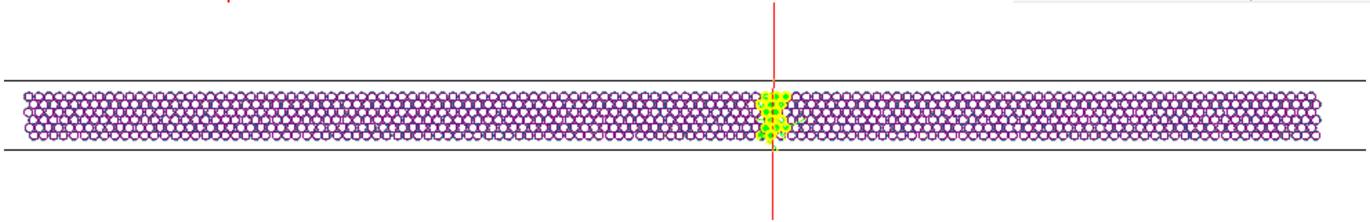
Blake Leverington

https://gitlab.cern.ch/bleverin/SciFiMatG4_v2




4 GeV
Muon

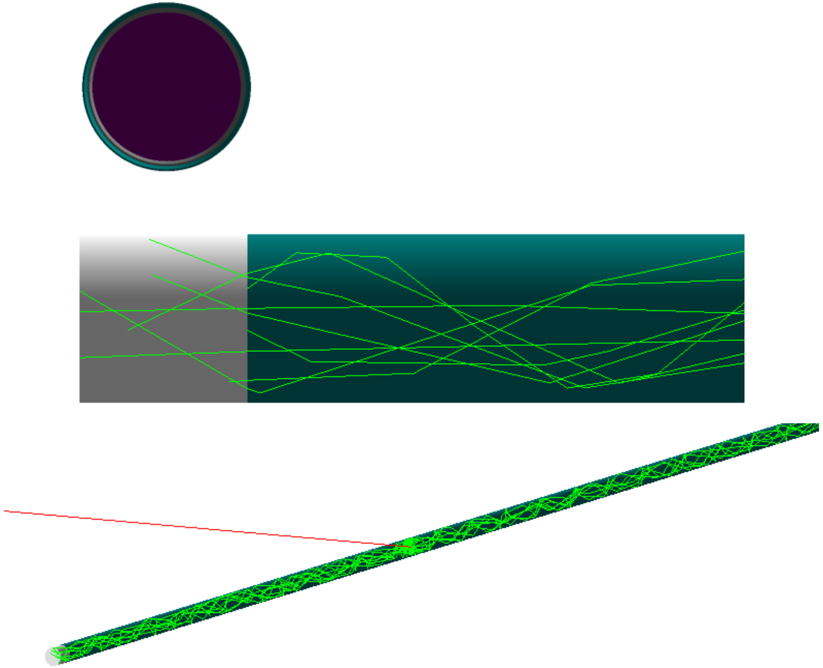




Extended code

Note: in CMakeLists.txt, change the line to enable OGL : option(WITH_GEANT4_UIVIS "Build example with Geant4 UI and Vis drivers" ON) instead of OFF

- Based on a single fibre simulation
  - https://gitlab.cern.ch/lhcb-scifi/SciFiSimG4
  - Which was based on Example N06 and LiXe



Analysis

Code snippet

- Several ROOT trees are used for storing the data from the tracks
  - **DetectedPhotons**: all the detected photons from the events
  - **Trigger**: Tracks that deposit energy in a Trigger volume
  - **EnergyTracks**: the energy and track length through the core of the fibre from charged particle
  - **PrimaryParticleTrack**: "primary particles position“
  - **InitialParticle**: "Initial Particle energy and momentum“
- You’ll see the Analysis class appear in the other files at different points.

```
void Analysis::FillTree(Double_t energy, Double_t time, Double_t length, Double_t absTime,
                        Double_t x, Double_t y, Double_t z,
                        Double_t px, Double_t py, Double_t pz,
                        Double_t vertexX, Double_t vertexY, Double_t vertexZ,
                        Double_t vertexPx, Double_t vertexPy, Double_t vertexPz,
                        Int_t trackId,
                        Int_t creatorProcess, Int_t parentId)
{
  if(detectedPhotonsTree!=NULL)
  {
    energyBuffer = energy;
    timeBuffer = time;
    lengthBuffer = length;

    wavelengthBuffer = 1239.842/energy;

    absTimeBuffer = absTime;
    xBuffer = x;
    yBuffer = y;
    zBuffer = z;

    pxBuffer = px;
    pyBuffer = py;
    pzBuffer = pz;

    vertexXBuffer = vertexX;
    vertexYBuffer = vertexY;
    vertexZBuffer = vertexZ;

    vertexPxBuffer = vertexPx;
    vertexPyBuffer = vertexPy;
    vertexPzBuffer = vertexPz;

    trackIdBuffer = trackId;

    creatorProcessBuffer = creatorProcess;
    parentIdBuffer = parentId;
```

Run Action

- A Run is a collection of generated primaries (events)

```
void RunAction::BeginOfRunAction(const G4Run* aRun)
{
    G4cout << "### Run " << aRun->GetRunID() << " start." << G4endl;
    timer->Start();
    Analysis::GetInstance()->PrepareNewRun(aRun);
    G4cout << "Analysis Instance Address: " << Analysis::GetInstance() << G4endl
;
}

void RunAction::EndOfRunAction(const G4Run* aRun)
{
    timer->Stop();
    G4cout << "number of event = " << aRun->GetNumberOfEvent() << " \t" << *time
r << G4endl;
    Analysis::GetInstance()->EndOfRun(/*aRun*/);
}
```


UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

Event Action

- An Event is a collection of tracks from each Primary generation

```
void EventAction::BeginOfEventAction(const G4Event* anEvent)
{
    G4cout << "# Event " << anEvent->GetEventID() << " start." << G4endl;
    Analysis::GetInstance()->PrepareNewEvent(anEvent);

    fEnergy = 0.0;
    fTrackL = 0.0;
}

void EventAction::EndOfEventAction(const G4Event* anEvent)
{
    G4double runID = (G4double) G4RunManager::GetRunManager()->GetCurrentRun()->i
GetRunID();
    Analysis::GetInstance()->FillEnergyTrack(runID,     anEvent->GetEventID(), fi
Energy, fTrackL);
}
```

Primary Generation

- I'm using a 3D histogram from another LHCb simulation as the input for the track angle and position
- Two* ways of generating primaries:
  - **General Particle Source** where you define source size and momentum distributions and they are randomly thrown
  - **Particle Gun** where you set the momentum of each particle specifically.
  - External particle physics generators such as Pythia for particle physics decays
- The particle type and energy can be left free to be set in the .mac file

```
PrimaryGeneratorAction::PrimaryGeneratorAction()
{
  // gun = InitializeGPS(); //general particle source
  gun = new G4ParticleGun(1); //particle gun
  rootfile = TFile::Open("distfile.root");
  disthist = (TH3F*)rootfile->Get("ath3_phi_theta_yHit;1");
}

void PrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{

 G4double yHit = 0*mm;
 G4double xHit = G4UniformRand()-0.5;
 disthist->GetRandom3(phi, theta, yHit); //in degrees and mm
 std::cout <<"Primary Phi:" << phi << "deg  Theta: " << theta << "deg  Y :" << yHit << "mm" << std::endl;
 G4int rndsign =  0;
 rndsign = (G4UniformRand()<0.5) ? 1 : -1 ;
 phi*=deg* rndsign;
 rndsign = (G4UniformRand()<0.5) ? 1 : -1 ;
 theta*=deg * rndsign;
 gun->SetParticlePosition(G4ThreeVector(xHit,2.0*mm,yHit+10.)); //Z here is Y in LHCb coordinates;
 // std::cout <<"Primary Phi:" << phi << "  Theta: " << theta << "  Y :" << yHit << std::endl;

 gun->SetParticleMomentumDirection(G4ThreeVector(sin(theta)*cos(phi), -cos(theta), sin(theta)*sin(phi)));
e away from verticle;


 gun->GeneratePrimaryVertex(anEvent);
```

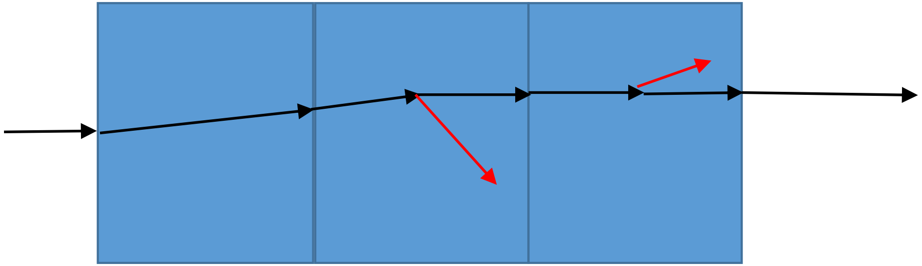
Stacking Action

- A “stack” of tracks generated during each event.
    - Each track is propagated one by one until the track is terminated for various reasons
- Secondaries are new particles (tracks) created from physical processes due to interactions of the primary particle and the materials or decays

```
G4ClassificationOfNewTrack StackingAction::ClassifyNewTrack(const G4Track * aTrack)
{
    /* ++ StackingAction for optical photons ++ */
    if(aTrack->GetDefinition() == G4OpticalPhoton::OpticalPhotonDefinition())
    {
        Analysis::GetInstance()->IncreaseReflectionRefractionAndScatteringVectors(aTrack->GetTrackID());
        Analysis::GetInstance()->IncreaseLengthVectors(aTrack->GetTrackID());

        if(aTrack->GetParentID()>0) // particle is secondary
        {
            gammaCounter++;
            G4int creatorProcessId = 0;

            if(aTrack->GetCreatorProcess()->GetProcessName() == "Scintillation")
                creatorProcessId = 1;

            if(aTrack->GetCreatorProcess()->GetProcessName() == "Cerenkov")
                creatorProcessId = 2;

            if(aTrack->GetCreatorProcess()->GetProcessName() == "OpWLS")
                creatorProcessId = 3;

            if (aTrack->GetCreatorProcess())
                AddProcess(aTrack->GetCreatorProcess());// classify here
        }
    }
    /* ++ End of StackingAction of optical photons ++ */
```

Stepping Action

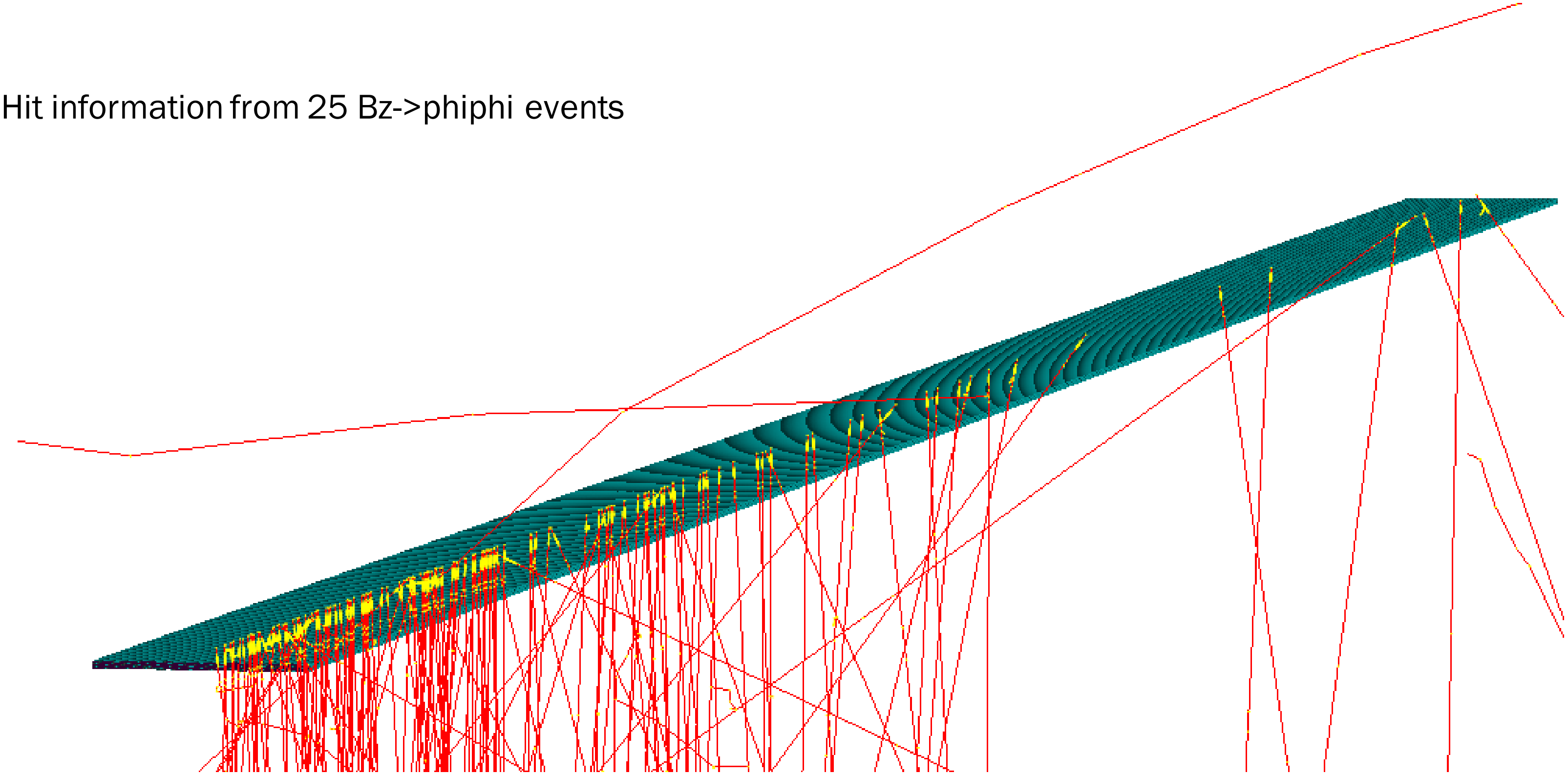
- The short steps of the particle tracks between material boundaries
- “smart” handling of tracks by killing them given certain conditions (saves CPU time)
- Analyse particle properties and processes from each step:
  - Number of reflections from surfaces
  - Energy deposited from charged particles in the scintillator
  - Distance through materials
  - Many more…



```
        // Count the reflections and refractions

        if(step->GetPostStepPoint()->GetStepStatus()==fGeomBoundary)
        {
            if(postPhysicalVolumeName == "EpoxyBox" && prePhysicalVolumeName.substr(0,9
) == "Cladding2")
            {
                if(Parameters::GetInstance()->ProbabilityOfPhotonLossAtSurface() == 1)
                    step->GetTrack()->SetTrackStatus(fStopAndKill);

                if(Parameters::GetInstance()->ProbabilityOfPhotonLossAtSurface() < 1
                   && Parameters::GetInstance()->ProbabilityOfPhotonLossAtSurface() > 0
)
                {
                    G4double randomNumber = CLHEP::RandFlat::shoot();

                    if(Parameters::GetInstance()->ProbabilityOfPhotonLossAtSurface() <
randomNumber)
                    {
                        if (step->GetStepLength() > 0)
                            Analysis::GetInstance()->IncreaseReflectionsAtFibreSurface(
trackId);
                    }
                    else
                        step->GetTrack()->SetTrackStatus(fStopAndKill);
                }

                if (step->GetStepLength() > 0)
                    Analysis::GetInstance()->IncreaseReflectionsAtFibreSurface(trackId)
;
            }
            if(postPhysicalVolumeName == "Mirror")
                Analysis::GetInstance()->IncreaseReflectionsAtMirror(trackId);


            // Retrieve status of the photon boundary process
```

Particle gun with parameter distributions from 3D ROOT Histogram (theta, phi, y).

Hit information from 25 Bz->phiphi events



The parameter file(s)

- Many (but not all) of the simulation parameters, emission and absoption spectra are read from parameter files defined in parameters.cc via the Parameter class
    - Some geometry parameters are defined here, some in DetectorConstruction.cc
- A summary file is output as well with descriptive text

```
 char parameterFileName[40] = "parameterFiles/parameters.dat";
 #pragma message("Compiling for Linux")
dif
 std::ifstream parameterFile;
 parameterFile.open(parameterFileName);

 if(parameterFile.good())
 {
     parameterFile >> randomSeed; // random seed for random engine
     parameterFile >> randomNumber; // random number for detector strip positioning
     randomNumber = randomNumber/32767.0; // max random number in bourne shell random number generator
     parameterFile >> fibreLength; // length of fibre in meter
     parameterFile >> semiAxisZ; // semi axis of fibre in z in millimeter
     parameterFile >> semiAxisY; // semi axis of fibre in y in millimeter
     parameterFile >> triggerX; // x-size of Trigger in mm
     parameterFile >> triggerY; // y-size of Trigger in mm
     parameterFile >> triggerZ; // z-size of Trigger in mm
     parameterFile >> triggerXPos; // x-position of Trigger in millimeter
     parameterFile >> triggerZPos; // z-position of Trigger in millimeter
     parameterFile >> probabilityOfPhotonLossAtSurface; // probability that a photon is killed when reaching fibre surface

     if(probabilityOfPhotonLossAtSurface<0 || probabilityOfPhotonLossAtSurface>1)
       probabilityOfPhotonLossAtSurface = 1;

     parameterFile >> placeMirror; // place a mirror at fibre end
     parameterFile >> mirrorReflectivity; // reflectivity of mirror at fibre end
     parameterFile >> detectorMaterial; // place a mirror at fibre end

     parameterFile.ignore(256,'\n');
     parameterFile.peek();
     parameterFile.getline(emissionSpectrumFileName,256);

     G4cout << "Emission FIle: " << emissionSpectrumFileName << G4endl;
```

DetectorConstruction.cc

Defining the detector geometry

DefineMaterials

- An LHCb Scintillating fibre mat has the following materials:
  - Scintillating Fibre
    - polystyrene core
    - PMMA inner cladding
    - f.PMMA outer cladding
  - Glue with TiO2
- An environment to place the materials in

Define a material with properties:

```
    // Elements to cunstruct outer cladding material (PTFEMA)
G4double densityPMMA2 = 1430*kg/m3;
std::vector<G4String> PMMA2_elm;
std::vector<G4int> PMMA2_nbAtoms;
PMMA2_elm.push_back("H"); PMMA2_nbAtoms.push_back(7);
PMMA2_elm.push_back("C"); PMMA2_nbAtoms.push_back(6);
PMMA2_elm.push_back("O"); PMMA2_nbAtoms.push_back(2);
PMMA2_elm.push_back("F"); PMMA2_nbAtoms.push_back(3);
PMMA2 = man->ConstructNewMaterial("PMMA2", PMMA2_elm, PMMA2_nbAtoms, densityPMMA2=1430*kg/m3);

// Outer cladding sections
G4String materialNameCladding2 = "OuterCladdingMaterial";
outerCladdingMaterial = new G4Material(materialNameCladding2,PMMA2->GetDensity(),1);
outerCladdingMaterial->AddMaterial(PMMA2,1.);
```

Use predefined materials in Geant4:

```
// Environment
Air     = man->FindOrBuildMaterial("G4_AIR");
Vacuum  = man->FindOrBuildMaterial("G4_Galactic");

// Polystyrene   G4_POLYSTYRENE
Pstyrene = man->FindOrBuildMaterial("G4_POLYSTYRENE");

// Core sections
G4String materialNameCore = "ScintCoreMaterial";
scintCoreMaterial = new G4Material(materialNameCore,Pstyrene->GetDensity(),1);
scintCoreMaterial->AddMaterial(Pstyrene,1.);
```

Scintillation in GEANT4

- A material will create Optical Photons from deposited ionisation energy
- Note: Material properties are attached to the process (and not the material).
- This means, at present, GEANT4 can only accommodate one scintillation material in any given application
- The yield may be set to be dependent on the type of primary particle.
  - Not implemented here yet (Geant4 v10.7 or greater needed)
  - Known saturation effects from ions

```
// Set scintillation and WLS properties

scintCoreMaterialProperties->AddProperty("FASTCOMPONENT",scintSpecVector);
scintCoreMaterialProperties->AddProperty("SLOWCOMPONENT",scintSpecVector);
scintCoreMaterialProperties->AddProperty("WLSCOMPONENT",wlsSpecVector);


scintCoreMaterialProperties->AddConstProperty("SCINTILLATIONYIELD",
                                              Parameters::GetInstance()->ScintillationYield()/keV);
scintCoreMaterialProperties->AddConstProperty("RESOLUTIONSCALE",
                                              Parameters::GetInstance()->ResolutionScale());
scintCoreMaterialProperties->AddConstProperty("FASTTIMECONSTANT",
                                              Parameters::GetInstance()->DecayTimeFast()*ns);
scintCoreMaterialProperties->AddConstProperty("SLOWTIMECONSTANT",
                                              Parameters::GetInstance()->DecayTimeSlow()*ns);
scintCoreMaterialProperties->AddConstProperty("YIELDRATIO", Parameters::GetInstance()->YieldRatio());
// Is set in "PhysicsList.hh" as well due to inconsistency in Geant4 (G4OpticalPhsysics default value)

scintCoreMaterialProperties->AddProperty("WLSABSLENGTH",
                                         WlsAbsEnergy,WlsAbsLength,WLS_ABS_ENTRIES)->SetSpline(true);

scintCoreMaterialProperties->AddConstProperty("WLSTIMECONSTANT", Parameters::GetInstance()->WlsDecayTime()*ns);
```

Optical Photons need material properties

- Absorption lengths are wavelength dependent
- **Radiation damage:** Implemented in SciFiSimG4 by using sections of Fibre with different optical properties (not included here in the Mat simulation for simplicity)

```
// Material properties table

G4MaterialPropertiesTable* scintCoreMaterialProperties = new G4MaterialPropertiesTable();
G4MaterialPropertiesTable* innerCladMaterialProperties = new G4MaterialPropertiesTable();
G4MaterialPropertiesTable* outerCladMaterialProperties = new G4MaterialPropertiesTable();


// Set absorption

// todo : write functions in c++ code
TF1 coreAbs("coreAbs",Parameters::GetInstance()->AbsorptionCore(),300,800);
TF1 clad1Abs("clad1Abs",Parameters::GetInstance()->AbsorptionClad1(),300,800);
TF1 clad2Abs("clad2Abs",Parameters::GetInstance()->AbsorptionClad2(),300,800);

G4double* Pstyrene_ABSLENGTH = new G4double[E_NUMENTRIES];
G4double* PMMA_ABSLENGTH = new G4double[E_NUMENTRIES];
G4double* PMMA2_ABSLENGTH = new G4double[E_NUMENTRIES];

// Calculate absorption lengths

for(int j=0; j<E_NUMENTRIES; j++)
{
    double wavelengthNanometer = Parameters::hcPERe/Energy[j]*1e3;

    Pstyrene_ABSLENGTH[j] = 1./coreAbs.Eval(wavelengthNanometer)*m;
    PMMA_ABSLENGTH[j]      = 1./clad1Abs.Eval(wavelengthNanometer)*m;
    PMMA2_ABSLENGTH[j]     = 1./clad2Abs.Eval(wavelengthNanometer)*m;
}

scintCoreMaterialProperties->AddProperty("ABSLENGTH",Energy,Pstyrene_ABSLENGTH,E_NUMENTRIES)->SetSpline(true);
innerCladMaterialProperties->AddProperty("ABSLENGTH",Energy,PMMA_ABSLENGTH,E_NUMENTRIES)->SetSpline(true);
outerCladMaterialProperties->AddProperty("ABSLENGTH",Energy,PMMA2_ABSLENGTH,E_NUMENTRIES)->SetSpline(true);
```

https://gitlab.cern.ch/lhcb-scifi/SciFiSimG4

Optical Photons need material properties

- Wavelength dependant refractive index

```
// Set refractive indices

scintCoreMaterialProperties->AddProperty("RINDEX",Energy,Pstyrene_RIND,E_NUMENTRIES)->SetSpline(true);
innerCladMaterialProperties->AddProperty("RINDEX",Energy,PMMA_RIND,E_NUMENTRIES)->SetSpline(true);
outerCladMaterialProperties->AddProperty("RINDEX",Energy,PMMA2_RIND,E_NUMENTRIES)->SetSpline(true);
```

```
// REFRACTIVE INDICES

G4double* Vacuum_RIND = new G4double[E_NUMENTRIES];
G4double* Pstyrene_RIND = new G4double[E_NUMENTRIES];
G4double* PMMA_RIND = new G4double[E_NUMENTRIES];
G4double* PMMA2_RIND = new G4double[E_NUMENTRIES];
G4double* Epoxy_RIND = new G4double[E_NUMENTRIES];

// Functions are saved in parameter-file
// todo: write this functions in c++ code
TF1 vacuumRind("vacuumRind",Parameters::GetInstance()->RefractiveIndexVacuum(),300,800);
TF1 coreRind("coreRind",Parameters::GetInstance()->RefractiveIndexCore(),300,800);
TF1 clad1Rind("clad1Rind",Parameters::GetInstance()->RefractiveIndexClad1(),300,800);
TF1 clad2Rind("clad2Rind",Parameters::GetInstance()->RefractiveIndexClad2(),300,800);

for(int i=0; i<E_NUMENTRIES; i++)
{
    double wavelengthNanometer = Parameters::hcPERe/Energy[i]*1e3;
    Vacuum_RIND[i] = vacuumRind.Eval(wavelengthNanometer);
    Pstyrene_RIND[i] = coreRind.Eval(wavelengthNanometer);
    PMMA_RIND[i] = clad1Rind.Eval(wavelengthNanometer);
    PMMA2_RIND[i] = clad2Rind.Eval(wavelengthNanometer);
    Epoxy_RIND[i] = 1.59;
}
```

Optical Photons need material properties

- Rayleigh scattering

```
// RAYLEIGH SCATTERING

G4double* Pstyrene_RAYLEIGH = new G4double[E_NUMENTRIES];
G4double* PMMA_RAYLEIGH = new G4double[E_NUMENTRIES];
G4double* PMMA2_RAYLEIGH = new G4double[E_NUMENTRIES];

TF1 coreRayleigh("coreRayleigh",Parameters::GetInstance()->RayleighCore(),300,800);
TF1 clad1Rayleigh("clad1Rayleigh",Parameters::GetInstance()->RayleighClad1(),300,800);
TF1 clad2Rayleigh("clad2Rayleigh",Parameters::GetInstance()->RayleighClad2(),300,800);

// Calculate scattering lengths

for(int j=0; j<E_NUMENTRIES; j++)
{
    double wavelengthNanometer = Parameters::hcPERe/Energy[j]*1e3;
    // todo : write this functions in c++
    Pstyrene_RAYLEIGH[j] = 1./coreRayleigh.Eval(wavelengthNanometer)*m;
    PMMA_RAYLEIGH[j]                = 1./clad1Rayleigh.Eval(wavelengthNanometer)*m;
    PMMA2_RAYLEIGH[j]       = 1./clad2Rayleigh.Eval(wavelengthNanometer)*m;
    }
```

Define the geometry

- After defining your materials, start with your World
  - Typically a big box of air or vacuum
- A physical Volume has:
  - a shape (GVBox, G4Tube, …)
  - A logical volume where you assign the shape, material, logicalName="World"
  - placement of a logical volume creates a physical volume with a location inside a parent.
    - World is our Top parent such that parent = 0
    - placementName

```
DefineMaterials();
DefineMaterialProperties();

// Placing the world (experimental hall)

 //------------------------------------------------
 // World
 //------------------------------------------------

 G4Box * world_box =
                   new G4Box("World", fWorldSizeX, fWorldSizeY, fWorldSizeZ);

 fLogicWorld = new G4LogicalVolume(world_box,
                                   Air,
                                   "World");

 fPhysiWorld = new G4PVPlacement(0,G4ThreeVector(0.,0.,0.), fLogicWorld,"World", 0, false, 0);
```

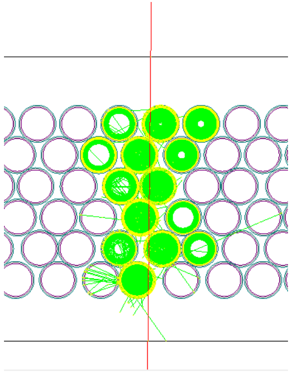
```
G4LogicalVolume::G4LogicalVolume ( G4VSolid *              pSolid,
                                   G4Material *            pMaterial,
                                   const G4String &        name,
                                   G4FieldManager *        pFieldMgr = 0,
                                   G4VSensitiveDetector *  pSDetector = 0,
                                   G4UserLimits *          pULimits = 0,
                                   G4bool                  optimise = true
                                 )
```

```
G4PVPlacement::G4PVPlacement ( G4RotationMatrix *        pRot,
                               const G4ThreeVector &     tlate,
                               G4LogicalVolume *         pCurrentLogical,
                               const G4String &          pName,
                               G4LogicalVolume *         pMotherLogical,
                               G4bool                    pMany,
                               G4int                     pCopyNo,
                               G4bool                    pSurfChk = false
                             )
```

The fibre mat

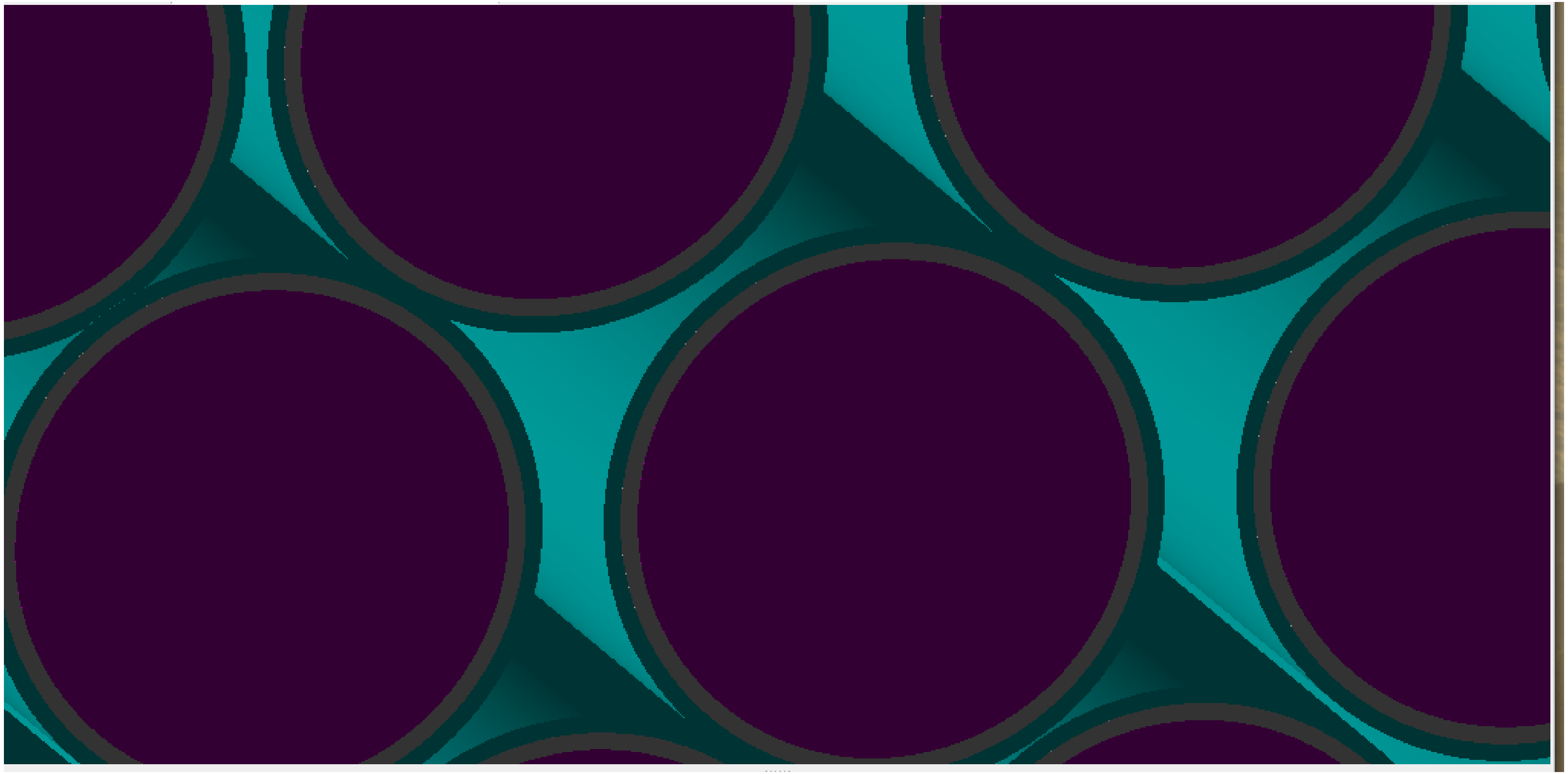
- A detector box for moving all the volumes around together inside the World volume

- The glue between the fibres is handled by placing the fibres inside a box of epoxy

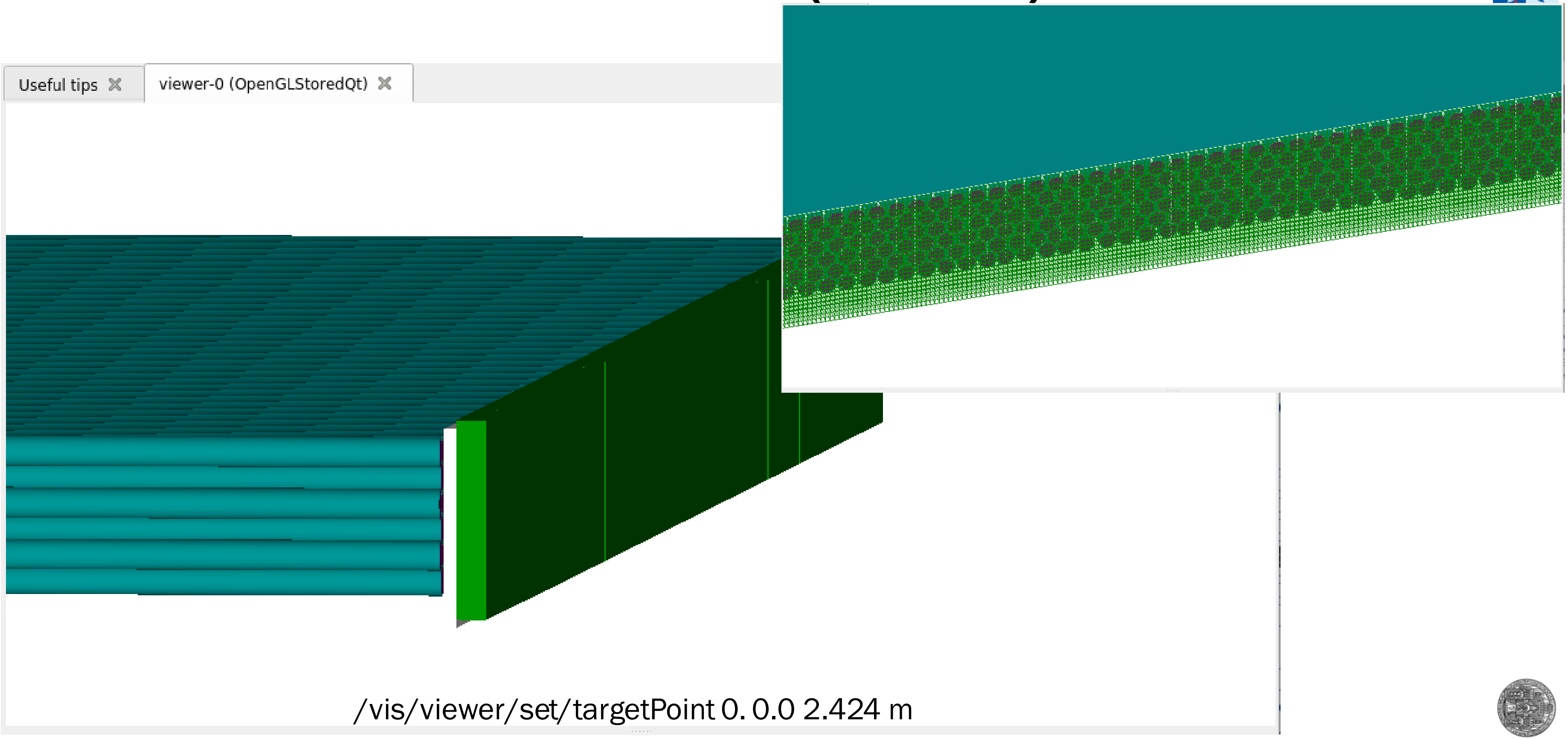


```
G4Box* detector_box = new G4Box("Detector",detector_x,detector_y,detector_z);
detector_log = new G4LogicalVolume(detector_box,Air,"Detector");
detector_phys = new G4PVPlacement(0,G4ThreeVector(0. , 0., scint_z ),detector_log,"Detector",fLogicWorld,false,0);


// Epoxy (a sheet of epoxy that the fibres are embedded inside)
G4double xEpoxy = 0.5*(Nk*xDist+2*Parameters::GetInstance()->SemiAxisZ()*mm)+5*mm;
G4double yEpoxy = 0.5*(Nj*yDist+2*Parameters::GetInstance()->SemiAxisZ()*mm)+0.2*mm;
G4Box* epoxyBox = new G4Box("EpoxyBox",xEpoxy, yEpoxy, scint_z);
epoxyLog = new G4LogicalVolume(epoxyBox, Epoxy, "EpoxyBox", 0, 0, 0);
epoxyPhy = new G4PVPlacement(0, G4ThreeVector() , epoxyLog, "EpoxyBox", detector_log, false, 0);


/* ++ Fibre Placement ++ */
// Outer cladding
G4Tubs* clad2Section_tube = new G4Tubs("Cladding2Section",clad2_rZmin,clad2_rZmax,dim_z, sphi,ephi);
G4LogicalVolume *clad2Section_log = new G4LogicalVolume(clad2Section_tube,
                                                        outerCladdingMaterial, "Cladding2Section",0,0,0);
// Inner cladding
G4Tubs* clad1Section_tube = new G4Tubs("Cladding1Section",clad1_rZmin,clad1_rZmax,dim_z, sphi,ephi);
G4LogicalVolume *clad1Section_log = new G4LogicalVolume(clad1Section_tube,
                                                        innerCladdingMaterial, "Cladding1Section",0,0,0);
// Scintillating core
G4Tubs* coreSection_tube = new G4Tubs("CoreSection",core_rZmin,core_rZmax,dim_z, sphi,ephi);
G4LogicalVolume *coreSection_log = new G4LogicalVolume(coreSection_tube,
                                                       scintCoreMaterial, "CoreSection",0,0,0);


for(int j = 0; j < Nj; j++)
{
    for(int k = 0; k < Nk; k++)
    {
        origin = objectPos(j,k);

        new G4PVPlacement(0, origin, clad2Section_log, "Cladding2"+C::c(j)+C::c(k), epoxyLog, true, 0);

        new G4PVPlacement(0, origin, clad1Section_log, "Cladding1"+C::c(j)+C::c(k), epoxyLog, true, 0);

        new G4PVPlacement(0, origin, coreSection_log, "Core"+C::c(j)+C::c(k), epoxyLog, true, 0);
    }
}
    /* ++ End of Fibre Placement ++ */
```

Handles the staggering of rows, random variations, etc



The Sensitive Detector (SiPMs)



The Sensitive Detector (SiPMs)

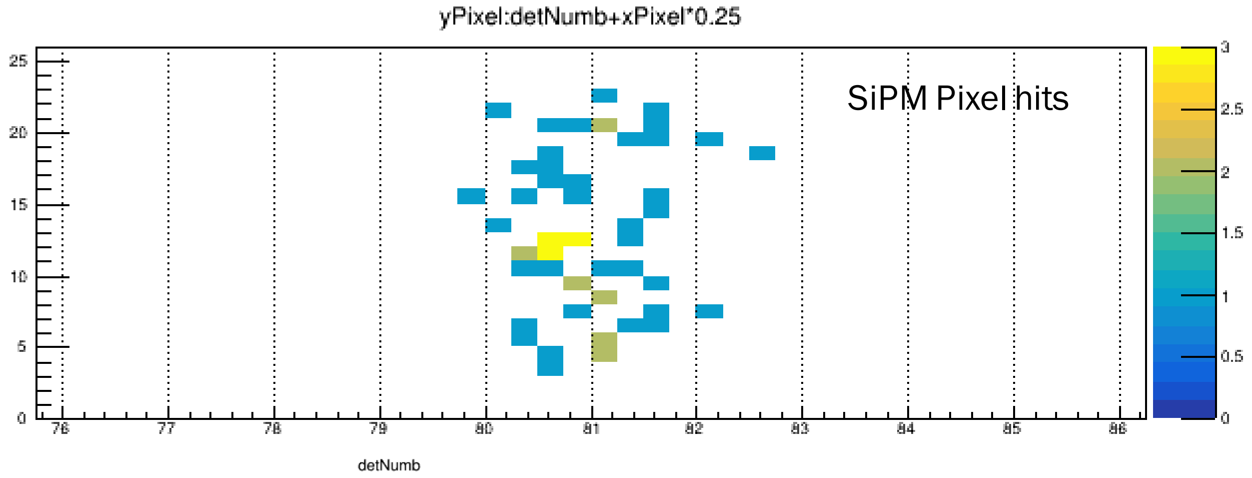
- A logical volume becomes "sensitive" if it has a pointer to a concrete class derived from **G4VSensitiveDetector**

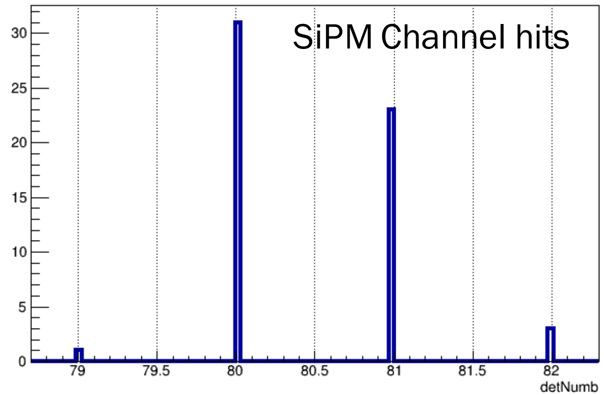
- You can create "Hits" which can then create a custom detector response when the particle enters the volume.
  - In this case, the Pixel is simply recorded in the DetectedPhotons tree and the photon is killed.

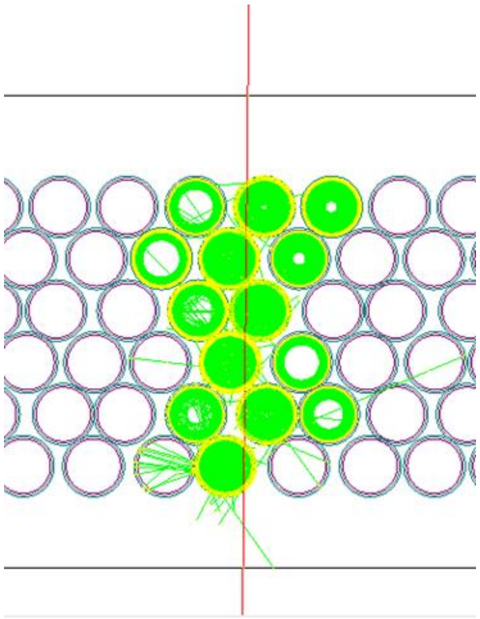
```
/* ++ Construction and placement of the detector strips ++ */

// Epoxy layer infront of det strip
G4VSolid* epoxy_strip = new G4Box("EpoxyStrip",  stripWidth/2., stripHeight/2., epoxy_strip_width/2.);
G4LogicalVolume* epoxy_strip_log = new G4LogicalVolume(epoxy_strip, Glue, "EpoxyStrip", 0, 0, 0);

//place the pixels at the end of the fibre mat
G4VSolid* pixelS = new G4Box("Pixel", pixelDimX/2., pixelDimY/2., stripWidth/2.);
G4LogicalVolume* pixelL = new G4LogicalVolume(pixelS, Glue, "Pixel", 0, 0, 0);

SensitiveDetector* sensitive = new SensitiveDetector("/Sensitive");
G4SDManager* sdman = G4SDManager::GetSDMpointer();
sdman->AddNewDetector(sensitive);
pixelL->SetSensitiveDetector(sensitive);

G4int Nk_s = ceil(((G4double)Nk)*(xDist/stripWidth)); // Determ. autom. nb. of detector strips.
for(int k = 1; k < Nk_s-1; k++)
{
    new G4PVPlacement(0, objectPos(Nj/2, k, scint_z+epoxy_strip_width/2.+airGap), epoxy_strip_log,
                      "EpoxyStrip", detector_log, false, 0);

    for(int i = 0; i < Nx; i++)
    {
        for(int j = 0; j < Ny; j++){
            new G4PVPlacement(0, objectPos(Nj/2, i, j, k, scint_z+stripWidth/2.+epoxy_strip_width+airGap),
                              pixelL, "SensitiveDetector"+C::c1(k)+C::c2(i)+C::c3(j), detector_log, false, 0);
        }
    }
}
    /* ++ End of Constr. and placem. of det. strips ++ */
```

https://geant4-userdoc.web.cern.ch/UsersGuides/ForApplicationDeveloper/html/Detector/hit.html?highlight=sensitive

ROOT output outFile_1.root







To prevent optical photon generation

Comment out this section /* */ in **src/PhysicsList.cc**
For only tracking primary particles. Then recompile.

```
    /*
    // * Optical Physics
    G4OpticalPhysics* opticalPhysics = new G4OpticalPhysics();
    RegisterPhysics( opticalPhysics );

    // adjust some parameters for the optical physics
    opticalPhysics->SetWLSTimeProfile("exponential");

    opticalPhysics->SetScintillationYieldFactor(1.0);
    opticalPhysics->SetScintillationExcitationRatio(Parameters::GetInstance()->Y
ieldRatio());

    opticalPhysics->SetMaxNumPhotonsPerStep(100);
    opticalPhysics->SetMaxBetaChangePerStep(10.0);

    // opticalPhysics->SetTrackSecondariesFirst(true);
    */
}


void PhysicsList::ConstructParticle()
{
    // Constructs all paricles
    G4VModularPhysicsList::ConstructParticle();
}


void PhysicsList::SetCuts()
U:**-  PhysicsList.cc    36% L45    Git:master   (C++//l Abbrev)
```

Compiling and running the software.

Follow the README.me file, and after every change, from the SciFiMatG4_v2 folder:

- source setup.sh
- mkdir -p build //it is recommended to delete this folder if you have errors during compiliation
- cd build
- cmake ../SciFiSim
- cmake --build . // or make clean; make

Run from SimulationData folder

- cd ../SimulationData
- ../build/scifiSim // opens a GUI

Or

- ../build/scifisim muongun.mac //batch mode when a .mac file is specified

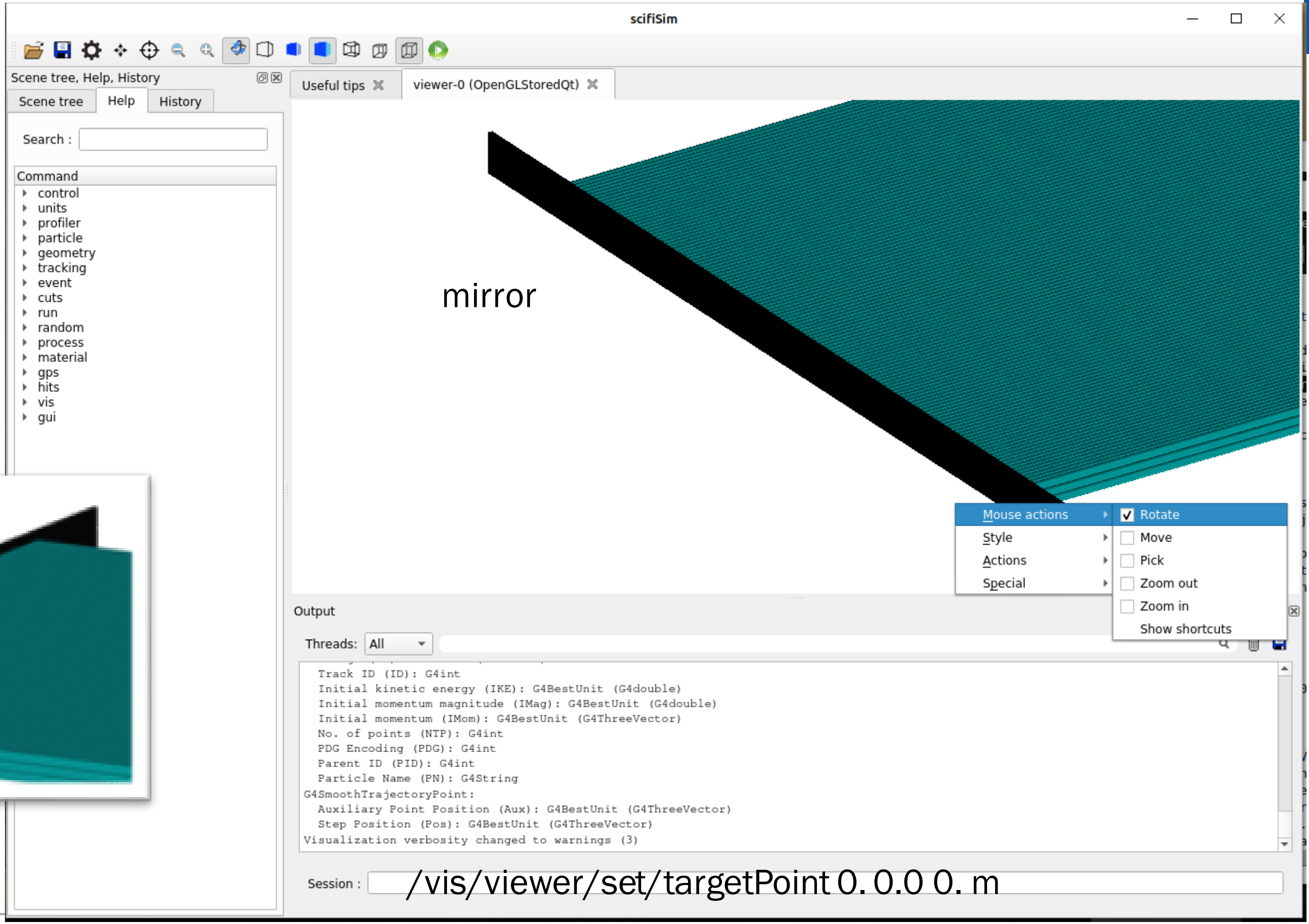
Use batch mode once you are satisfied with your setup and only need to collect results.

The visualization macro (vis.mac)

- Draw your volumes
- Hide some volumes
- Change view angles, colours, etc
- Generate primaries and visualize tracks
- Can all be done from the interface command line too

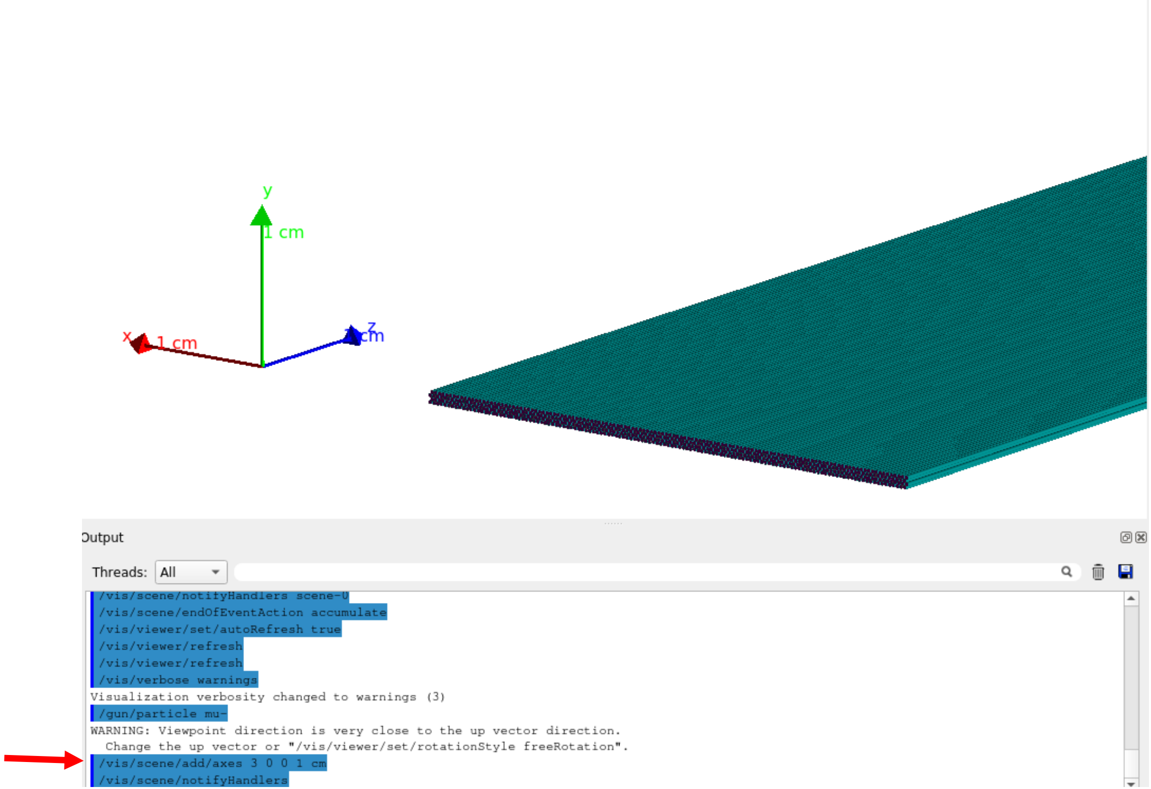
```
/control/verbose 1
/event/verbose 0
/run/verbose 2
/vis/open OGL 600x600-0+0
/vis/viewer/set/autoRefresh false
/vis/verbose errors


/vis/drawVolume
/vis/scene/add/trajectories smooth
/tracking/storeTrajectory 2


# To get nice view
# Make the "World" box invisible

/vis/geometry/set/visibility World 0 false
/vis/geometry/set/visibility Detector 0 false
/vis/geometry/set/visibility EpoxyBox 0 false
/vis/geometry/set/visibility AbsBox 0 false
/vis/geometry/set/visibility EpoxyStrip 0 false
/vis/geometry/set/visibility Pixel 0 false
/vis/geometry/set/visibility Trigger 0 false
/vis/geometry/set/visibility Mirror 0 false

/vis/geometry/set/colour all 0 0 0 0

/vis/geometry/set/colour Cladding2Section 1 0. 0.5 (
/vis/geometry/set/colour Cladding1Section 0 0.5 0.5
/vis/geometry/set/colour CoreSection 0 0.5 0.0 0.5
/vis/geometry/set/colour Pixel 0 0.0 0.5 0.0 1.0
/vis/geometry/set/colour EpoxyStrip 0 1.0 1.0 1.0 1
/vis/geometry/set/colour EpoxyBox 0 0.9 0.9 0.9 1.
#/vis/geometry/set/colour EpoxyBoxsub 0 0.25 0.25 0
```

```
# Specify style (surface, wireframe, auxiliary edges,...)
#/vis/viewer/set/style wireframe
#/vis/viewer/set/auxiliaryEdge true
/vis/viewer/set/style surface
#edge is slow
/vis/viewer/set/lineSegmentsPerCircle 100
#/vis/viewer/set/edge false
#/vis/viewer/set/hiddenMarker true
/vis/viewer/set/targetPoint 0.0 0.0 .0 m
#slight angle view from mirror end
#/vis/viewer/set/viewpointThetaPhi 120 50
/vis/viewer/set/viewpointThetaPhi 180 90
/vis/viewer/zoomTo 20
/vis/viewer/set/background white
/vis/viewer/set/projection p


/vis/scene/notifyHandlers
/vis/modeling/trajectories/create/drawByCharge
/vis/modeling/trajectories/drawByCharge-0/default/setDrawStepPts true
/vis/scene/notifyHandlers scene-0
/vis/modeling/trajectories/drawByCharge-0/default/setStepPtsSize 2
/vis/scene/notifyHandlers scene-0


/vis/scene/endOfEventAction accumulate


/vis/viewer/set/autoRefresh true
/vis/viewer/refresh
/vis/verbose warnings
```


scifiSim
Scene tree, Help, History
Scene tree
Help
History
Search :
Command
control
units
profiler
particle
geometry
tracking
event
cuts
run
random
process
material
gps
hits
vis
gui
Useful tips
viewer-0 (OpenGLStoredQt)
mirror
Mouse actions
Style
Actions
Special
Rotate
Move
Pick
Zoom out
Zoom in
Show shortcuts
Output
Threads: All
Track ID (ID): G4int
Initial kinetic energy (IKE): G4BestUnit (G4double)
Initial momentum magnitude (IMag): G4BestUnit (G4double)
Initial momentum (IMom): G4BestUnit (G4ThreeVector)
No. of points (NTP): G4int
PDG Encoding (PDG): G4int
Parent ID (PID): G4int
Particle Name (PN): G4String
G4SmoothTrajectoryPoint:
Auxiliary Point Position (Aux): G4BestUnit (G4ThreeVector)
Step Position (Pos): G4BestUnit (G4ThreeVector)
Visualization verbosity changed to warnings (3)
Session :
/vis/viewer/set/targetPoint 0. 0.0 0. m


mirror

Add axes


y
1 cm
x
1 cm
z
cm
Output
Threads: All
/vis/scene/notifyHandlers scene-0
/vis/scene/endOfEventAction accumulate
/vis/viewer/set/autoRefresh true
/vis/viewer/refresh
/vis/viewer/refresh
/vis/verbose warnings
Visualization verbosity changed to warnings (3)
/gun/particle mu-
WARNING: Viewpoint direction is very close to the up vector direction.
Change the up vector or "/vis/viewer/set/rotationStyle freeRotation".
/vis/scene/add/axes 3 0 0 1 cm
/vis/scene/notifyHandlers