Tracking

Introduction

Tracking in Geant4 is independent of

- **Particle Type**
- **Physics Process**

Depending upon the process cross-section, it contibutes in following ways

- **Determining the step length**
- **Generate secondary particles**
- **Suggests changes in the state of particles**

When a process generates secondary particles, these secondary particles are pushed onto the stack.

Once the current track is completely processed, its secondaries are popped out from stack for further processing.

Overview of Tracking in Geant4

- Tracking follows particles from creation to termination step-by-step.
- - Includes interactions, decays, and field effects.
- - Each step contributes to simulation accuracy.
- - Handles secondary particle generation.

The Tracking Algorithm

- 1. Generate primary particle.
- 2. Initialize track.
- 3. Calculate steps based on interactions.
- 4. Terminate track when conditions met.
- 5. Track secondary particles.

Geant4 Classes for Tracking

- - **G4Track** : Stores particle properties.
- - **G4Step :** Describes movement.
- - **G4Trajectory :** Stores full path.
- - **G4VProcess :** Handles interactions.

Complete workflow in Geant4
Run->Event->Tracks->Steps

Run

Collection of Events

/run/beamOn n

Once a run start, whole simulation environment is freezed and can't be changed.

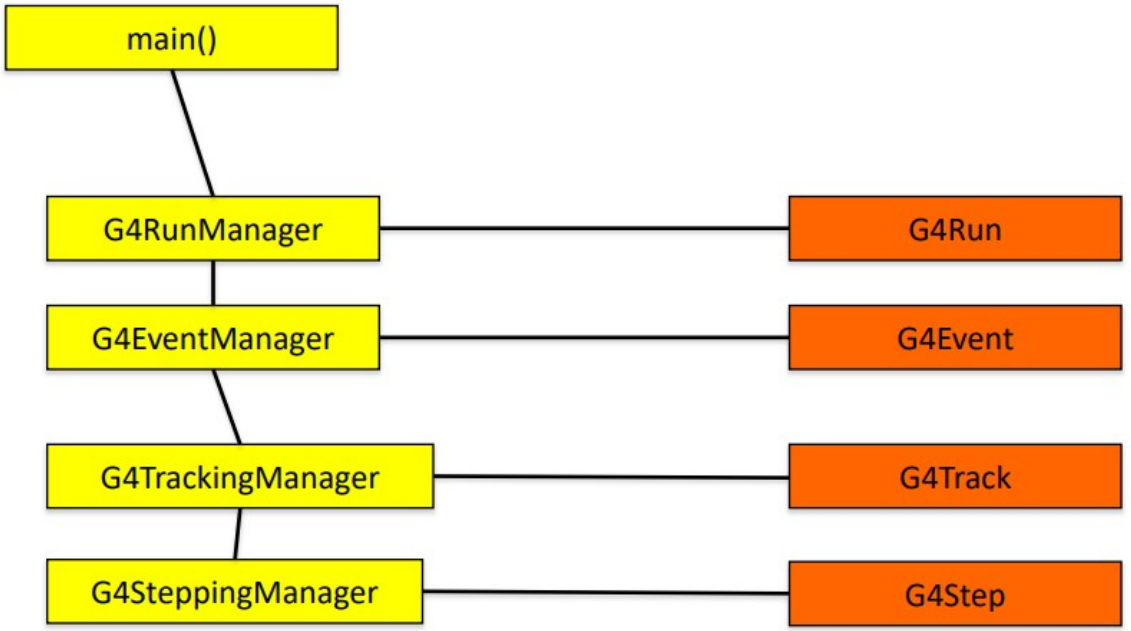
-->Detector setup

--> Particles, Processes, etc.

Represented by G4Run Class

To get the information about the Run, the G4UserRunAction hook class can be used (optional).

Good place to connect resources like open a file to store the data.



Event

Basic Unit

Represents the complete lifetime starting from generation of primary particle and termination of primary as well as all the secondary particle produced as a results of various interactions.

At the beginning primary tracks are generated and pushed into a stack.

A track is popped from the stack one by one and tracked by navigator.

The resultant secondary track are pushed in the stack.

The tracking continues as long as ther are tracks present in the stack.

Event cont..

An Event is controlled by G4Event class.

The class is managed by G4EventManager class.

G4UserEventAction is the optional user hook, which user can use to interact with the Geant4 Event.

Consist of two important virtual funtions:

(a) BeginOfEventAction (Correct location to allocate resources which are local to an event)

(b) EndOfEventAction (Correct location to send information collected in an event to Run, e.g. total energy deposited in a volume in an event)

Getting information from EventAction by intercepting EventAction class

```
class G4UserEventAction
{

      G4UserEventAction();
      virtual ~G4UserEventAction();
      virtual void BeginOfEventAction(const
       G4Event* anEvent);
      virtual void EndOfEventAction(const
       G4Event* anEvent);
}
```

```
class MyClass_EventAction : public
G4UserEventAction{

    MyClass_EventAction();
    ~MyClass_EventAction();
     Doubel eDep;

    void BeginOfEventAction(const G4Event* anEvent){
       //Write your stuff here
       //Initialize all event related parameter
       eDep=0;
    }
    void EndOfEventAction(const G4Event* anEvent){
       //Write your stuff here
       //Print total energy deposited
       //Use G4RunManager::GetRunManager()
     }
}
;
```

Now just register the object of your **EventAction** with RunManager

```
SetUserAction(new MyClass_EventAction);
```

What is a Track ??

Our perception : A sequence of steps followed by the particle

In Geant4 terminology sequence of steps is a Trajectory, not a track.

Track : Represent by G4Track class represent the current state (snapshot of a particle).

Gives information about the particle at the time of query.

Pointer to the track object can be obtained from steps by call GetTrack() function.

G4TrackingManager is the manager class

G4UserTrackingAction is the optional user hook

Create your own Tracking class inherited from G4UserTrackingAction to implement new features, and to interact with Track during the simulation

Getting information from Trackingaction

```
class G4UserTrackingAction
{

      G4UserTrackingAction();
      virtual ~G4UserTracking();
      virtual void
PreUserTrackingAction(const
G4Track* track);
      virtual void
PostUserTrackingAction(const
G4Track *track);
}
```

```
class MyClass_TrackingAction : public
G4UserTrackingAction{

    MyClass_TrackingAction();
    ~MyClass_TrackingAction();
     Doubel eDep;

    void PreUserTrackingAction(const G4Track* track){
       //Write your stuff here
       //Get all the secondaries generated
       //Filter secondaries

    }
    void PostUserTrackingAction(const G4Track* track){
       //Write your stuff here
       //Process the complete trajectory

     }
}
```

Now just register the object of your **EventAction** with RunManager

```
SetUserAction(new MyClass_EventAction);
```

Lifetime of a Track Object

Each Track object disappears (is deleted) when it either

– leaves the outermost (‘world’) volume,

– disappears in an interaction (e.g. by decay or inelastic scattering),

– it’s kinetic energy becomes zero

– the user decides to kill it (‘artificially’).

– all the Tracks (particles) disappears in the end of event.

Situation where UserTrackingAction is useful !!!

Holds a pointer to G4TrackingManager

Contains 2 important function

1) PreUserTrackingAction(const G4Track*) :

Called when a new track is about to be processed and useful for following

Filtering or classifying track based on certain condition (eg. Store only charged particles..)

Enabling or disabling trajectory storage for specific particles.

[By default Geant4 does not store the trajectory to save memory and performance, but it can be done if required.]

If Geant4 does not store trajectory, how it visualizes them ?

During interactive mode, Geant4 dynamically creates and visualizes trajectories.

This behavior is controlled by the visualization manager and trajectory drawing settings.

[Try running your simulation in interactive mode and batch mode]

These objects are passed to the visualization manager, which renders them in OpenGL, HepRep, or other visualization engines.

The temporary trajectories exist only while the event is processed, and they are discarded after visualization.

If you want to save them: You must explicitly enable storage using SetStoreTrajectory(true).

[Do it carefully, as you are going to store a lot of data]

Mode	Are Trajectories Shown?	Are Trajectories Stored?
Interactive Mode	Yes (temporary for visualization)	No (unless `SetStoreTrajectory(true)`)
Batch Mode	No (not even temporary)	No (unless explicitly enabled)

Trajectory verification

Lets verify the trajectory in the interactive and batch mode.

Can be done by querying the TrackingManager in PreUserTrackingAction() function

```
void MyTrackingAction::PreUserTrackingAction(const G4Track* track) {
   if (fpTrackingManager->GetStoreTrajectory()) {
      G4cout << "Trajectory is being stored for Track ID: "
           << track->GetTrackID() << G4endl;
   } else {
      G4cout << "No trajectory storage for Track ID: "
           << track->GetTrackID() << G4endl;
   }
}
```

Things to Try :

Run you code in (1) Batch mode (2) Interactive mode.

Now force trajectory storage : fpTrackingManager->SetStoreTrajectory(true);

Again Run you code in (1) Batch mode (2) Interactive mode.

Some Important points about Trajectory

Remember that you need to query TrackingManager object fpTrackingManager in order to get the trajectory.

Three important function related to TrackingManager

(1) GetStoreTrajectory() (2) SetStoreTrajectory()

(3) GimmeTrajectory()

GetStoreTrajectory() just tell whether the trajectory storage is enable or not.

It does not give the Trajectory itself.

GimmeTrajectory() function actually gives you a trajectory only if trajectory storage flag is set.

PostUserTrackingAction(const G4Track*)

Called when track has finished processing.

Storing or analyzing track data.

Collecting secondary particles information : counting number of secondaries produces as a result of interaction.

TrackingManager keeps the tracks of creations of all the particles. GimmeSecondaries() function of tracking manager may be called to get the number of secondary particles produced.

Have a look the the source code G4UserTrackingAction.hh

Small Quiz

What is the correct location to call these following functions

GetStoreTrajectory()

SetStoreTrajectory()

GimmeTrajectory()

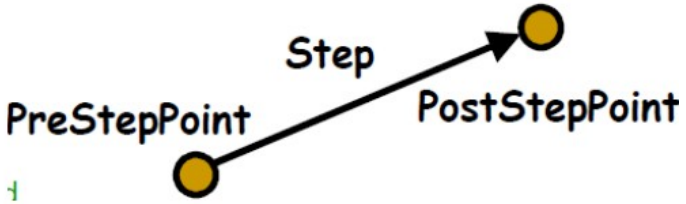
Step in Geant4

An event consists of number of trajectories of primary and secondary particles.

A trajectory consist of of number of steps.

Step is represented by G4Step class, and consist of two point and the delta information within these two point.

Does not store particle information, instead it maintains a pointer to G4Track class.



It basically stores delta information. Information about the particle at between two points.

Point is represented by G4StepPoint class

G4SteppingManager is manager class responsible for managing track and can give the pointer to the current step.

G4UserSteppingAction is the optional user hook class that can be used to interact with the current step.

Step Selection

In Geant4, transportation is a process that is associated with each particle type.

The transport process interacts with geometry boundaries.

All the processes associated with a particle participates for step selection and proposed a step length.

The process with the minimum step length becomes the winner, and the particle is moved by the step length proposed by the winner process.

The proposed step length given by a process is judged by the interaction length of the particle in the volume medium.

Scoring using UserSteppingAction

UserSteppingAction class of Geant4 allows to access information from individual steps

SteppingManager class manages each and every step.

To access the steps user needs to write a its stepping action class which should be inherited from G4UserSteppingAction class

Needs to override a function UserSteppingAction(G4Step *step) to get the required information from the step.

Internally the Geant4 engine calls this function at every step.

Not a mandatory function.

But becomes mandatory when you want the information control in your program.

Getting information from SteppingAction

```
class G4UserSteppingAction
{
   G4UserSteppingAction();
   virtual ~G4UserSteppingAction();

   virtual void UserSteppingAction(const
G4Step*){;}
};
```

```
class MyClass_SteppingAction : public
G4UserSteppingAction{

   Sim01_SteppingAction();
   ~Sim01_SteppingAction();

   void UserSteppingAction(const G4Step *step){
      //Write your stuff here like
      //Use G4RunManager::GetRunManager()

      std::cout << step->GetLength() << std::endl;
      std::cout << step->GetTotalEnergyDeposit() <<
std::endl;
   }
};
```

Now just register the object of your **SteppingAction class** with RunManager
SetUserAction(new MyClass_SteppingAction);

Quiz

Suppose you fire gammas of 662 keV on NaI detector

How to get the total energy deposited in a detector in an event.

Exercise of hands-on session

Try to match the tracking output obtained using stepping action with the output that you get from */tracking/*verbose 1

Generate the energy spectra from NaI detector

Information from Tracking verbose

```
**************************************************************************************************
* G4Track Information:   Particle = gamma,   Track ID = 1,   Parent ID = 0
**************************************************************************************************

Step#    X(mm)    Y(mm)    Z(mm) KinE(MeV)  dE(MeV) StepLeng TrackLeng  NextVolume ProcName
    0        0        0      -60     0.662        0        0         0       World initStep
    1        0        0    -27.5     0.662        0     32.5      32.5 PhysicalNaiCrystalCasingEndCap Tra
nsportation
    2        0        0    -26.5     0.662        0        1      33.5       World Transportation
    3        0        0    -26.5     0.662        0    1e-05      33.5 PhysicalNaiCrystal Transportation
    4        0        0    -22.3     0.662        0     4.21      37.7 PhysicalNaiCrystal Rayl
    5     2.21    -1.09     1.81     0.276 3.64e-05     24.2      61.9 PhysicalNaiCrystal compt
    6       24    -15.4    -1.18     0.276        0     26.2      88.2       World Transportation
    7     24.4    -15.7    -1.24     0.276        0    0.503      88.7 PhysicalNaiCrystalCasing Transport
ation
    8     25.2    -16.3    -1.35     0.276        0     1.01      89.7       World Transportation
    9      500     -329    -66.8     0.276        0      573       662  OutOfWorld Transportation

**************************************************************************************************
* G4Track Information:   Particle = e-,   Track ID = 2,   Parent ID = 1
**************************************************************************************************

Step#    X(mm)    Y(mm)    Z(mm) KinE(MeV)  dE(MeV) StepLeng TrackLeng  NextVolume ProcName
    0     2.21    -1.09     1.81     0.386        0        0         0 PhysicalNaiCrystal initStep
    1     2.21    -1.11     1.84     0.363   0.0229   0.0427    0.0427 PhysicalNaiCrystal msc
    2     2.23    -1.16     1.84     0.327   0.0356    0.117     0.159 PhysicalNaiCrystal eIoni
    3     2.25    -1.13      1.8     0.254   0.0729    0.103     0.262 PhysicalNaiCrystal eIoni
    4     2.27    -1.13     1.77     0.222   0.0321   0.0752     0.337 PhysicalNaiCrystal eIoni
    5     2.28    -1.15      1.8     0.198   0.0246    0.064     0.401 PhysicalNaiCrystal eIoni
    6     2.29    -1.16     1.82     0.169   0.0287   0.0559     0.457 PhysicalNaiCrystal eIoni
    7      2.3    -1.17     1.82     0.151   0.0183   0.0469     0.504 PhysicalNaiCrystal eIoni
    8      2.3    -1.17      1.8     0.112   0.0381   0.0415     0.545 PhysicalNaiCrystal eIoni
    9      2.3    -1.18      1.8    0.0388   0.0736   0.0313     0.577 PhysicalNaiCrystal eIoni
   10      2.3    -1.18      1.8    0.0201   0.0187   0.0129     0.589 PhysicalNaiCrystal eIoni
   11      2.3    -1.18      1.8         0   0.0201   0.0045     0.594 PhysicalNaiCrystal eIoni
```

Messenger

Detector needs to know about Messenger.

Messenger needs to know about Detector.

Inherit you class from G4UImessenger