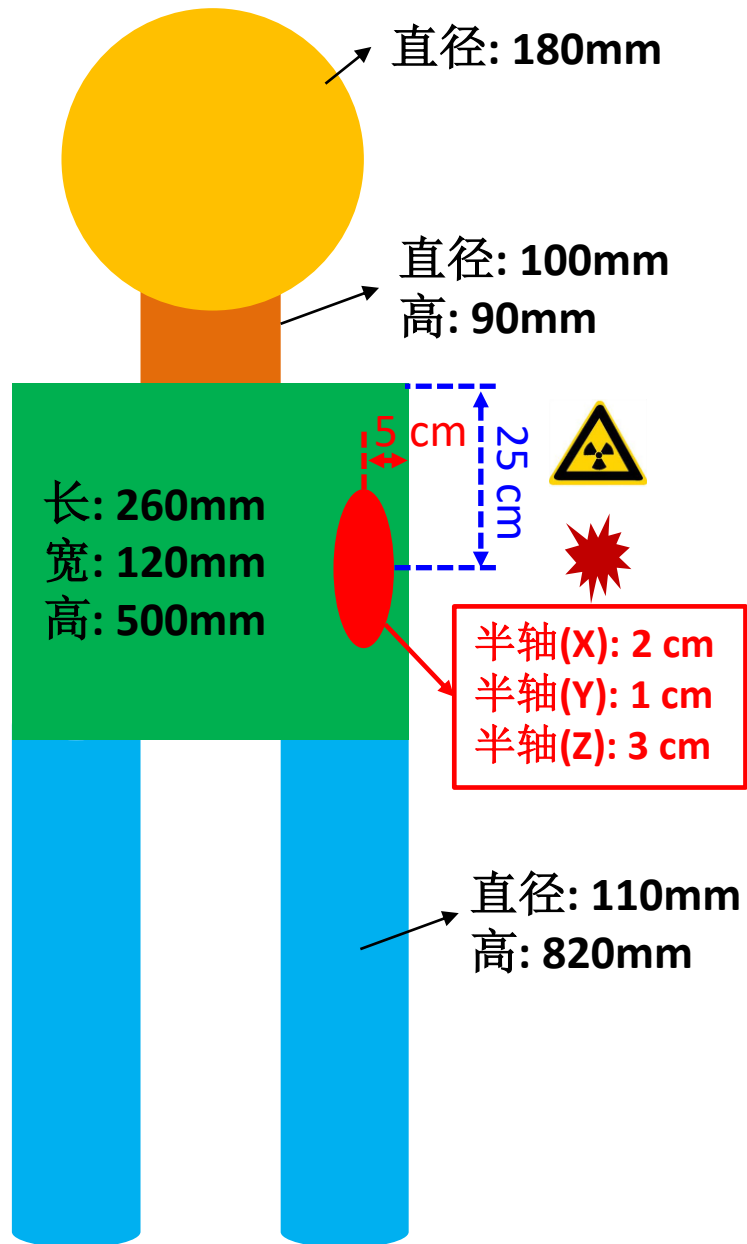


探测器模拟与数据分析

--探测器模拟基础

曹国富

中国科学院高能物理研究所



- ❖ 放疗是肿瘤治疗的一种常用手段，射线种类有伽马/质子/中子/离子等。
- ❖ 请设计并实现一个探测器模拟程序，从能量沉积的大小、尺度和位置几方面，对伽马和质子射线能量进行优化，对比使用这两种射线进行放疗的优劣。
- ❖ 硼中子俘获疗法(BNCT)是一种新的疗法，2020年全球首个BNCT设备在日本获批上市。假如使用0.5 eV的热中子束流，请研究预期治疗效果与肿瘤区内 ^{10}B 原子浓度的关系，以及束流截面大小的影响。最后，与其它射线的治疗效果进行对比。
- ❖ 肿瘤大小和位置如图红色区域所示。

❄ **Run, event, track, step, step point**

❄ **Track \leftrightarrow trajectory, step \leftrightarrow trajectory point**

❄ **Process**

➤ **At rest, along step, post step**

❄ **Cut = production threshold**

- ❄ **As an analogy of the real experiment, a run of Geant4 starts with “Beam On”**
- ❄ **Within a run, the user cannot change**
 - **detector setup**
 - **settings of physics processes**
- ❄ **Conceptually, a run is a collection of events which share the same detector and physics conditions**
 - **A run consists of one event loop.**
- ❄ **At the beginning of a run, geometry is optimized for navigation and cross-section tables are calculated according to materials appear in the geometry and the cut-off values defined**
- ❄ **G4RunManager class manages processing a run, a run is represented by G4Run class or a user-defined class derived from G4Run**
 - **A run class may have a summary results of the run**
- ❄ **G4UserRunAction is the optional user hook**

- ❄ **An event is the basic unit of simulation in Geant4**
- ❄ **At beginning of processing, primary tracks are generated. These primary tracks are pushed into a stack**
- ❄ **A track is popped up from the stack one by one and “tracked”. Resulting secondary tracks are pushed into the stack**
 - **This “tracking” lasts as long as the stack has a track**
- ❄ **When the stack becomes empty, processing of one event is over**
- ❄ **G4Event class represents an event. It has following objects at the end of its (successful) processing**
 - **List of primary vertices and particles (as input)**
 - **Hits and Trajectory collections (as output)**
- ❄ **G4EventManager class manages processing an event**
- ❄ **G4UserEventAction is the optional user hook**

❄ **Track is a snapshot of a particle**

- It has physical quantities of current instance only. It does not record previous quantities
- **Step** is a “delta” information to a track. Track is not a collection of steps. Instead, a track is being updated by steps

❄ **Track object is deleted when**

- it goes out of the world volume,
- it disappears (by e.g. decay, inelastic scattering),
- it goes down to zero kinetic energy and no “AtRest” additional process is required, or
- the user decides to kill it artificially

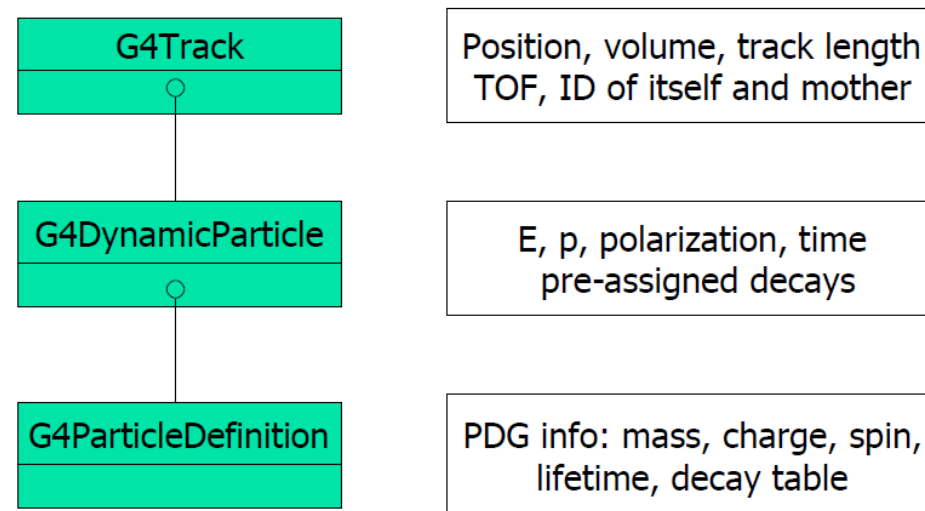
❄ **No track object persists at the end of event**

- For the record of tracks, use trajectory class objects

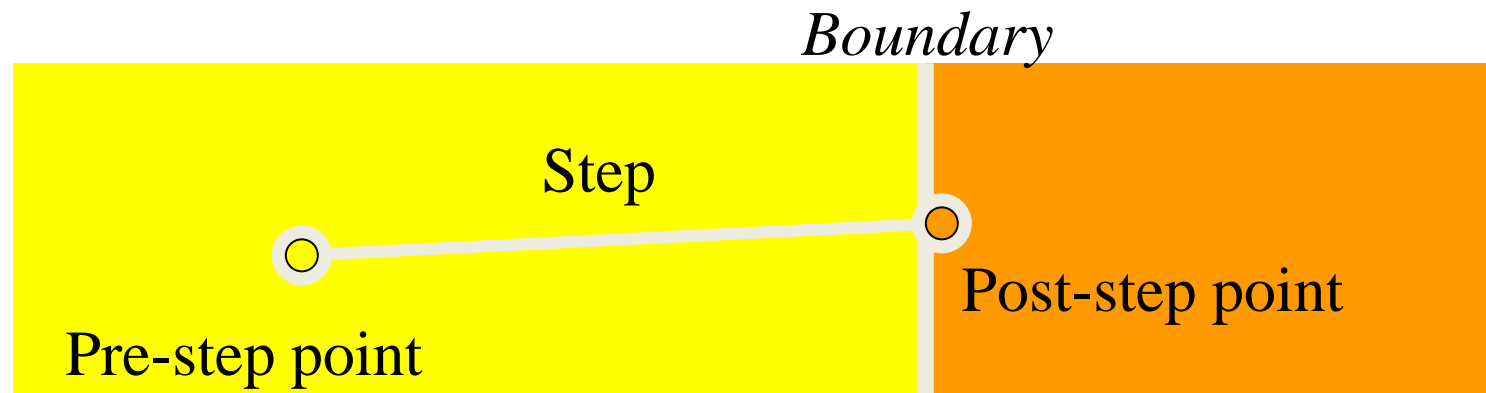
❄ **G4TrackingManager manages processing a track, a track is represented by G4Track class**

❄ **G4UserTrackingAction is the optional user hook**

- ❄ **A particle in Geant4 is represented by three layers of classes**
- ❄ **G4Track**
 - This is a class representing a particle to be tracked
- ❄ **G4DynamicParticle**
 - Each G4Track object has its own and unique G4DynamicParticle object
 - This is a class representing an individual particle
- ❄ **G4ParticleDefinition**
 - G4ProcessManager which describes processes involving to the particle
 - All G4DynamicParticle objects of same kind of particle share the same G4ParticleDefinition

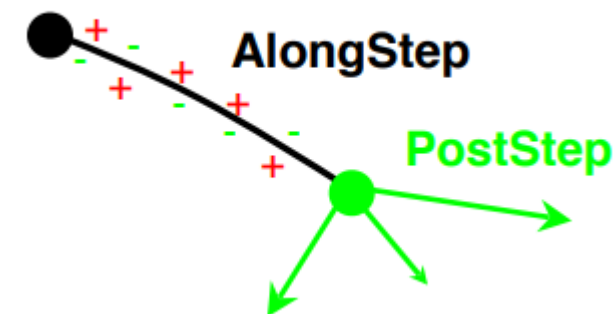


- ❄ **Step** has two points and also “delta” information of a particle (energy loss on the step, time-of-flight spent by the step, etc.)
- ❄ **Each point knows the volume (and material). In case a step is limited by a volume boundary, the end point physically stands on the boundary, and it logically belongs to the next volume**
 - Because one step knows materials of two volumes, boundary processes such as transition radiation or refraction could be simulated
- ❄ **G4SteppingManager** class manages processing a step, a step is represented by **G4Step** class
- ❄ **G4UserSteppingAction** is the optional user hook



- ❄ **Track does not keep its trace. No track object persists at the end of event**
- ❄ **G4Trajectory is the class which copies some of G4Track information. G4TrajectoryPoint is the class which copies some of G4Step information**
 - G4Trajectory has a vector of G4TrajectoryPoint.
 - At the end of event processing, G4Event has a collection of G4Trajectory objects.
 - /tracking/storeTrajectory must be set to 1.
- ❄ **Keep in mind the distinction**
 - $G4Track \leftrightarrow G4Trajectory$, $G4Step \leftrightarrow G4TrajectoryPoint$
- ❄ **Given G4Trajectory and G4TrajectoryPoint objects persist till the end of an event, you should be careful not to store too many trajectories**
 - E.g. avoid for high energy EM shower tracks.
- ❄ **G4Trajectory and G4TrajectoryPoint store only the minimum information**
 - You can create your own trajectory / trajectory point classes to store information you need. G4VTrajectory and G4VTrajectoryPoint are base classes.

- ❄ **All the work of particle decays and interactions is done by processes**
 - Transportation is also handled by a process
- ❄ **A process does two things:**
 - Decides when and where an interaction will occur
 - Method: `GetPhysicalInteractionLength()`
 - Generates the final state (changes momentum, generates secondaries, etc)
 - Method: `DoIt()`
- ❄ **The process which requires the shortest interaction length limits the step**
- ❄ **Each process has one or combination of the following natures:**
 - Well-located in space → **PostStep** → **Discrete process**
 - e.g. decay on the fly, hadronic interaction, ...
 - Not well-located in space → **AlongStep** → **Continuous process**
 - e.g. Cerenkov process, ionisation, ...
 - Well-located in time → **AtRest** → **AtRest process**
 - e.g. muon decay at rest



- ❄ **At the end of each step, according to the processes involved, the state of a track may be changed**
 - The user can also change the status in **UserSteppingAction**
 - Statuses shown in **green** are artificial, i.e. Geant4 kernel won't set them, but the user can set
- ❄ **fAlive**
 - Continue the tracking
- ❄ **fStopButAlive**
 - The track has come to zero kinetic energy, but still AtRest process to occur
- ❄ **fStopAndKill**
 - The track has lost its identity because it has decayed, interacted or gone beyond the world boundary
 - Secondaries will be pushed to the stack
- ❄ **fKillTrackAndSecondaries**
 - Kill the current track and also associated secondaries
- ❄ **fSuspend**
 - Suspend processing of the current track and push it and its secondaries to the stack
- ❄ **fPostponeToNextEvent**
 - Postpone processing of the current track to the next event
 - Secondaries are still being processed within the current event

❄ **Step status is attached to G4StepPoint to indicate why that particular step was determined.**

- Use “**PostStepPoint**” to get the status of this step.
- “**PreStepPoint**” has the status of the previous step.

❄ **fWorldBoundary**

- Step reached the world boundary

❄ **fGeomBoundary**

- Step is limited by a volume boundary except the world

❄ **fAtRestDoItProc, fAlongStepDoItProc, fPostStepDoItProc**

- Step is limited by a AtRest, AlongStep or PostStep process

❄ **fUserDefinedLimit**

- Step is limited by the user Step limit

❄ **fExclusivelyForcedProc**

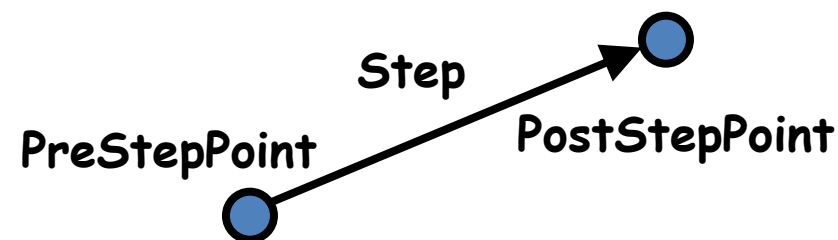
- Step is limited by an exclusively forced (e.g. shower parameterization) process

❄ **fUndefined**

- Step not defined yet

❄ **If you want to identify the first step in a volume, pick fGeomBoundary status in PreStepPoint.**

❄ **If you want to identify a step getting out of a volume, pick fGeomBoundary status in PostStepPoint.**

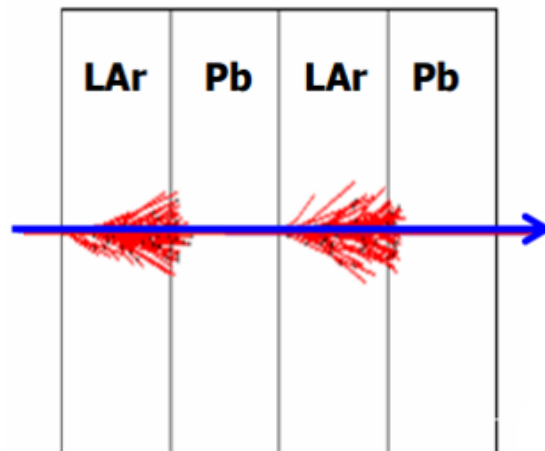


❄ Geant4 does not have tracking cuts

- All tracks are traced down to **zero** kinetic energy (by default).
- Can be implemented **manually** by user

❄ Production cuts (W_0)

- To decide whether a secondary particle to be produced or not.
- Applied only to: γ from **bremsstrahlung**, e^- from **ionization** and low- energy **protons** from **hadronic elastic scattering**
- This cut is a **distance**, and internally converted to energy threshold (W_0)
 - So, different materials have different W_0
- Particles unable to travel range cut distance will not be produced.



Production in range = 1.5 mm

455 keV electron energy in liquid Ar
2 MeV electron energy in Pb

- ❄ **Given geometry, physics and primary track generation, Geant4 does proper physics simulation “silently”**
 - You have to add a bit of code to **extract information useful to you**

- ❄ **There are two ways:**
 - Use user hooks (G4UserTrackingAction, G4UserSteppingAction, etc.)
 - You have an access to almost all information.
 - Straight-forward, but do-it-yourself
 - Use Geant4 scoring functionality
 - Assign **G4VSensitiveDetector** to a volume
 - **Hits collection** is automatically stored in G4Event object, and automatically accumulated if **user-defined Run** object is used.
 - Use user hooks (**G4UserEventAction**, **G4UserRunAction**) to get event / run summary

- ❄ **Very common way for a large project**
 - **Highly customizable, save whatever you want**
- ❄ **Create a class of SD and register it to the SD Manager**
- ❄ **Assign the pointer of your SD to the logical volume of your detector geometry**
- ❄ **Must be done in ConstructSDandField() of the user geometry**

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

G4SDManager* sdMan =G4SDManager::GetSDMpointer();

sdMan->AddNewDetector(mySensitive);

SetSensitiveDetector("LVname",mySensitive); // or

//logicVol->SetSensitiveDetector(mySensitive);
```

- ❄ Create a class of Hit, and define the variables that you want to save
- ❄ Should be derived from G4VHit
- ❄ The HitCollection is automatically managed by Geant4, and can be accessed in user hooks
- ❄ Add hits to HitCollection in your SD
- ❄ Refer to [examples/basic/B2/B2a](#)

```
class myHit : public G4VHit {  
    public:  
    ...  
    const myHit& operator=(const myHit&);  
    G4int operator==(const myHit&) const;  
  
    void SetTrackID (G4int track) { fTrackID = track; };  
    G4int GetTrackID() const { return fTrackID; };  
    private:  
    G4int fTrackID;  
};  
typedef G4THitsCollection<myHit> myHitsCollection;
```

```
G4bool mySD::ProcessHit(G4Step* astep) {  
    myHit* newHit = new myHit();  
    newHit->SetTrackID (aStep->GetTrack()->GetTrackID());  
    fHitsCollection->insert( newHit );  
    return true;  
};  
void mySD::EndOfEvent(G4HCofThisEvent*)  
{  
    G4int nofHits = fHitsCollection->entries();  
    for ( G4int i=0; i<nofHits; i++ )  
        (*fHitsCollection)[i]->Print();  
}
```

- ❄ **Geant4 provides a number of primitive scorers, each one accumulating one physics quantity (e.g. total dose) for an event.**
- ❄ **G4MultiFunctionalDetector is a concrete class derived from G4VSensitiveDetector**
 - It should be assigned to a logical volume as a kind of sensitive detector
 - It takes an arbitrary number of G4VPrimitiveScorer classes, to define the scoring quantities that you need
 - Each G4VPrimitiveScorer accumulates one physics quantity for each physical volume
 - E.g. G4PSDoseScorer (a concrete class of G4VPrimitiveScorer provided by Geant4) accumulates dose for each cell
 - Each G4VPrimitiveScorer is automatically named as **<MultiFunctionalDetectorName>/<PrimitiveScorerName>**
- ❄ **By using this approach, no need to implement sensitive detector and hit classes!**

```
MyDetectorConstruction::ConstructSDandField()
{
    G4MultiFunctionalDetector* myScorer = new
    G4MultiFunctionalDetector("myCellScorer");
    myCellLog->SetSensitiveDetector(myScorer);
    G4VPrimitiveScorer* totalSurfFlux = new
        G4PSFlatSurfaceFlux("TotalSurfFlux");
    myScorer->RegisterPrimitive(totalSurfFlux);
    G4VPrimitiveScorer* totalDose = new
        G4PSDoseDeposit("TotalDose");
    myScorer->RegisterPrimitive(totalDose);
}
```

instantiate multi-functional detector

attach to volume

create a primitive scorer (surface flux) and register it

create a primitive scorer (total dose) and register it

❄ **Concrete Primitive Scorers (→ Application Developers Guide 4.4.5)**

➤ **Track length**

- **G4PSTrackLength, G4PSPassageTrackLength**

➤ **Deposited energy**

- **G4PSEnergyDeposit, G4PSDoseDeposit**

➤ **Current/Flux**

- **G4PSFlatSurfaceCurrent,**
- **G4PSSphereSurfaceCurrent, G4PSPassageCurrent,**
- **G4PSFlatSurfaceFlux, G4PSCellFlux, G4PSPassageCellFlux**

❄ **Others**

- **G4PSMinKinEAtGeneration, G4PSNofSecondary, G4PSNofStep, G4PSCellCharge**

❄ **A G4VSDFilter can be attached to G4VPrimitiveScorer to define which kind of tracks have to be scored (e.g. one wants to know surface flux of **protons** only)**

- **G4SDChargeFilter** (accepts only **charged** particles)
- **G4SDNeutralFilter** (accepts only **neutral** particles)
- **G4SDKineticEnergyFilter** (accepts tracks in a defined range of **kinetic energy**)
- **G4SDParticleFilter** (accepts tracks of a given **particle type**)
- **G4VSDFilter** (base class to create user-customized filters)

```
MyDetectorConstruction::ConstructSDandField()  
{  
    G4VPrimitiveScorer* protonSurfFlux  
    = new G4PSFlatSurfaceFlux("pSurfFlux");  
    G4VSDFilter* protonFilter = new  
        G4SDParticleFilter("protonFilter");  
    protonFilter->Add("proton");  
  
    protonSurfFlux->SetFilter(protonFilter);  
  
    myScorer->RegisterPrimitive(protonSurfFlux);  
}
```

create a primitive
scorer (**surface
flux**), as before

create a **particle
filter** and add
protons to it

register the **filter**
to the primitive
scorer

register the **scorer** to the
multifunc detector (as
shown before)

```
//needed only once
G4int collID = G4SDManager::GetSDMpointer()
    ->GetCollectionID("myCellScorer/TotalSurfFlux");
G4HCofThisEvent* HCE = event->GetHCofThisEvent();
G4THitsMap<G4double>* evtMap =
    static_cast<G4THitsMap<G4double>*>
    (HCE->GetHC(collID));
for (auto pair : *(evtMap->GetMap())) {
    G4double flux = *(pair.second);
    G4int copyNb = *(pair.first);
}
```

Get **ID** for the collection (given the name)

Get **all HC** available in this event

Get the HC with the **given ID** (need a cast)

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

```
//needed only once
G4int collID = G4SDManager::GetSDMpointer()
    ->GetCollectionID("myCellScorer/TotalSurfFlux");
G4HCofThisEvent* HCE = event->GetHCofThisEvent();
G4THitsMap<G4double>* evtMap =
    static_cast<G4THitsMap<G4double>*>
    (HCE->GetHC(collID));
```

Get **ID** for the collection (given the name)

Get **all HC** available in this event

Get the HC with the **given ID** (need a cast)

	HCofThisEvent	
	Scorer 1	Scorer 2
Event#1	(0, 5.32)	<div style="border: 1px dashed black; padding: 5px; display: inline-block;"> (0, 1.43) (2, 7.41) </div>