Experimental Techniques in Particle Physics

Geant4:
User Action

Andreas Nowack
nowack@physik.rwth-aachen.de
RWTH Aachen University

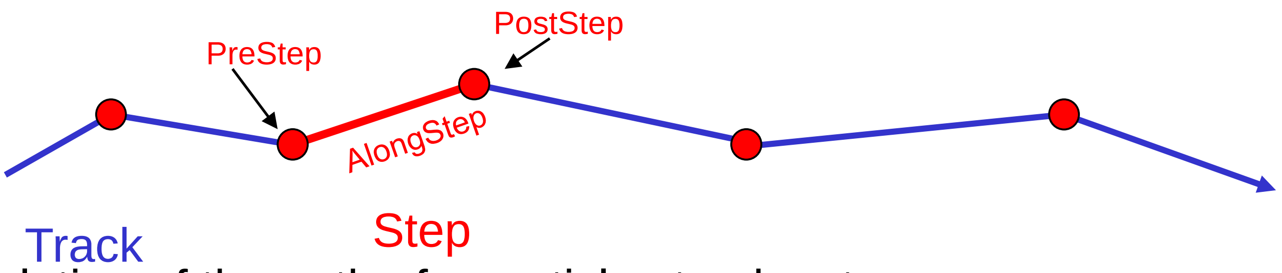
WS 2020/21

Quick Intro to
Geant 4

Processes

Recap

- each particle has its own G4ProcessManager providing a list of processes that this particle can undertake



- simulation of the path of a particle step by step

- three possibilities for processes to take place
  - at rest G4VProcess::AtRestDoIt
  - along step G4VProcess::AlongStepDoIt
  - post step G4VProcess::PostStepDoIt

- DoIt methods of the process class performs the physics processes:
  - momentum change
  - production of secondary particles

Actions in Geant4

User classes

Initialization classes

Invoked at the initialization:

G4VUserDetectorConstruction → Material and Geometry

G4VUserPhysicsList → Particles and Processes

Action classes

Invoked during an event loop:

G4VUserPrimaryGeneratorAction → Primary Particles

G4UserRunAction

G4UserEventAction

G4UserSteppingAction

Today's Tutorial

G4UserStackingAction

G4UserTrackingAction

main()

Geant4 does not provide main().

Note: classes written in Red are mandatory.

Different Levels of User Actions

- **Run**
  - at beginning of the run, e.g.:
    - initialize an analysis tool (root)
    - open/initialize input/output files
  - at end of the run, e.g.:
    - finish analysis tool
    - close opened files
- **Event**
  - at beginning of the event, e.g.
    - get parameters of the event
    - initialize variables used to analyse the event
  - at end of event, e.g.:
    - finish analysis of event
- **Stepping**
  - at every step of a particle’s trajectory, e.g.:
    - determine energy loss at every step

UserRunAction

UserEventAction

UserSteppingAction

Mandatory User Initializations

```
#include "G4RunManager.hh"
#include "DetectorPhysDetectorConstruction.hh"
#include "DetectorPhysPhysicsList.hh"
#include "DetectorPhysPrimaryGeneratorAction.hh"
```

in main():

```
 // Construct the default run manager
G4RunManager* runManager = new G4RunManager;

// set mandatory initialization classes
DetectorPhysDetectorConstruction* detector = new DetectorPhysDetectorConstruction;
runManager->SetUserInitialization(detector);

G4VUserPhysicsList* the_physics = new DetectorPhysPhysicsList;
runManager->SetUserInitialization(the_physics);

DetectorPhysPrimaryGeneratorAction* primarygeneration = new DetectorPhysPrimaryGeneratorAction(detector);
runManager->SetUserAction(primarygeneration);

// Initialize G4 kernel
runManager->Initialize();
```

Optional User Initializations

```
#include "DetectorPhysRunAction.hh"
#include "DetectorPhysEventAction.hh"
#include "DetectorPhysSteppingAction.hh"
```

in main():

```
DetectorPhysPrimaryGeneratorAction *primarygeneration = new DetectorPhysPrimaryGeneratorAction(detector);
runManager->SetUserAction(primarygeneration);

DetectorPhysRunAction* runaction = new DetectorPhysRunAction;
runManager->SetUserAction(runaction);

DetectorPhysEventAction* evtaction = new DetectorPhysEventAction(runaction);
runManager->SetUserAction(evtaction);

DetectorPhysSteppingAction* stepaction = new DetectorPhysSteppingAction(evtaction, runaction);
runManager->SetUserAction(stepaction);

// Initialize G4 kernel
runManager->Initialize();
```

Class “G4UserRunAction”

- header file:

```
class DetectorPhysRunAction : public G4UserRunAction {
  public:
    DetectorPhysRunAction();
    virtual ~DetectorPhysRunAction();

    virtual void BeginOfRunAction(const G4Run*);
    virtual void EndOfRunAction(const G4Run*);
};
```

- source file:

```
DetectorPhysRunAction::DetectorPhysRunAction() {}
DetectorPhysRunAction::~DetectorPhysRunAction() {}

// Begin of Run Action
void DetectorPhysRunAction::BeginOfRunAction(const G4Run* aRun) {
  G4cout << "Start of Run" << G4endl;
}

// End of Run Action
void DetectorPhysRunAction::EndOfRunAction(const G4Run* aRun) {
  G4cout << "End of Run" << G4endl;
}
```

Class “G4UserEventAction”

- header file:

```
class DetectorPhysEventAction : public G4UserEventAction {
  public:
    DetectorPhysEventAction(DetectorPhysRunAction*);
    virtual ~DetectorPhysEventAction();

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

- source file:

```
DetectorPhysEventAction::DetectorPhysEventAction(DetectorPhysRunAction* DetectorPhysRA) {}
DetectorPhysEventAction::~DetectorPhysEventAction() {}

// Begin of Event Action
void DetectorPhysEventAction::BeginOfEventAction(const G4Event* evt) {
  G4cout << "Start of Event" << G4endl;
}

// End of Event Action
void DetectorPhysEventAction::EndOfEventAction(const G4Event* evt) {
  G4cout << "End of Event" << G4endl;
}
```

Class “G4UserSteppingAction”

- header file:

```
class DetectorPhysSteppingAction : public G4UserSteppingAction {
  public:
    DetectorPhysSteppingAction(DetectorPhysEventAction*, DetectorPhysRunAction*);
    virtual ~DetectorPhysSteppingAction();

    virtual void UserSteppingAction(const G4Step* step );

  private:
    DetectorPhysEventAction* eventaction;
    DetectorPhysRunAction* runaction;
};
```

- source file:

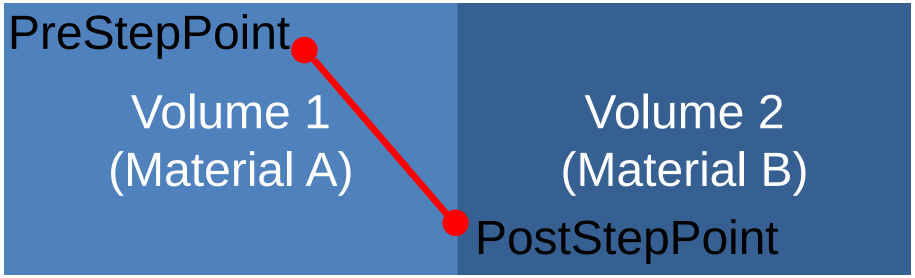
```
DetectorPhysSteppingAction::DetectorPhysSteppingAction(DetectorPhysEventAction* EA,
                                                       DetectorPhysRunAction* RA)
  : eventaction(EA), runaction(RA) {}

DetectorPhysSteppingAction::~DetectorPhysSteppingAction() {}

void DetectorPhysSteppingAction::UserSteppingAction(const G4Step* aStep) {
  // add whatever you like, examples see below
}
```

Steps in Geant4

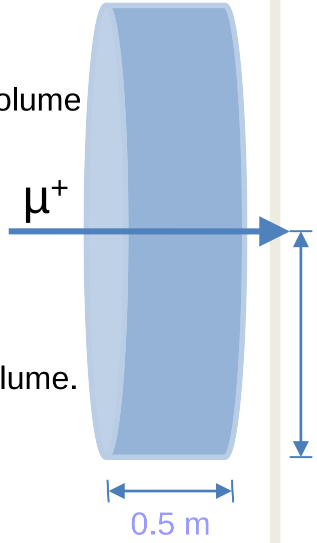
- Step:
  - two points (PreStepPoint and PostStepPoint)
    - each point knows its volume and thus its material
    - if step is limited by a volume boundary:
      - end point is at the boundary and logically belongs to the next volume
    - simulation of boundary processes such as transition radiation or refraction
  - “delta” information of a particle
    - energy loss on the step
    - time of flight spent by the step
    - …



Exercise: Energy Loss in Target

1. Download DetectorPhys_T8.tar.gz and decompress it.
2. Define a water material using NIST database.
3. Compile your code.
4. Construct a cylindrical target with 1 m radius and 0.5 m height (thickness).
5. Fill this target with the water material.
6. Compile your code.
7. Edit the macro file vis_T8.mac:
    1. Choose μ^+ with 1 GeV as your primary particle
    2. Shoot one particle onto your target.
8. Run your code.
9. Add User Run Action and edit DetectorPhysRunAction.cc:
    - In order to understand Geant4 during the execution, display (“G4cout”) which run number (ID) has started and ended.
10. Compile your code and run it. Make sure that you see what you would like to display.
11. Add User Event Action and edit DetectorPhysEventAction.cc:
    - Display which event number (ID) has started and ended.
12. Compile your code and run it. Make sure that you see what you would like to display.

13. **Is the energy deposited by a charged particle equal to the energy loss of this particle?**
    1. Add User Stepping Action and edit DetectorPhysSteppingAction.cc.
    2. Display the step number and the volume name where the step is.
    3. Display the name of the particle doing this step.
    4. Determine its kinetic energy, … in the pre- and post-step points.
    5. Determine the name of the next volume.



Useful Methods

See also Geant4's class reference

G4Run* aRun
G4Event* evt
G4Step* aStep
already defined in the code

aRun->GetRunID() ⇒ G4int

evt->GetEventID() ⇒ G4int

aStep->GetStepLength() ⇒ G4double

aStep->GetPreStepPoint()->GetKineticEnergy() ⇒ G4double

aStep->GetPostStepPoint()->GetKineticEnergy() ⇒ G4double

aStep->GetTrack()->GetDynamicParticle()->GetDefinition()->GetParticleName() ⇒ G4String

aStep->GetTrack()->GetCurrentStepNumber() ⇒ G4int

aStep->GetTrack()->GetDefinition()->GetPDGCharge() => G4double

aStep->GetTrack()->GetTrackID() ⇒ G4int

aStep->GetTrack()->GetParentID() ⇒ G4int

aStep->GetTrack()->GetVolume()->GetName() ⇒ G4String

if (aStep->GetTrack()->GetNextVolume())
    aStep->GetTrack()->GetNextVolume()->GetName() ⇒ G4String

aStep->GetTotalEnergyDeposit() => G4double