More on Kernel

I. Hrivnacova, IJCLab Orsay

Credits: M. Asai (SLAC)

Geant4 IN2P3 and ED PHENIICS Tutorial,
22 – 26 May 2022, IJCLab

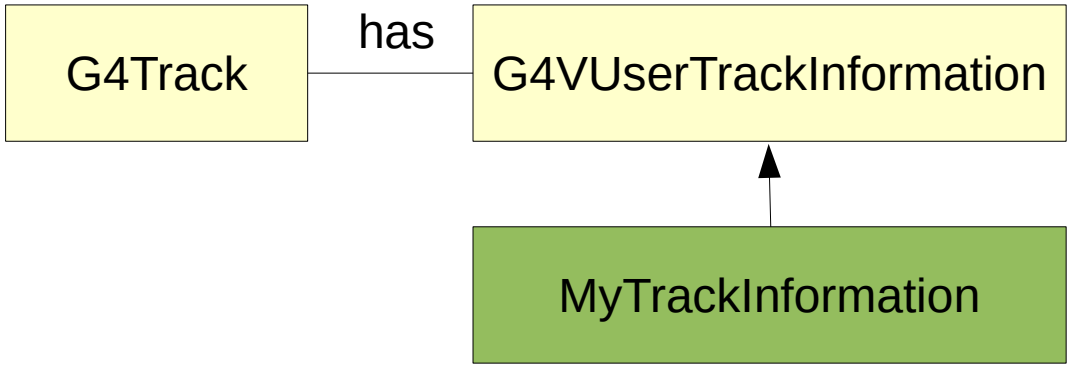
Outline

- User Information classes, User run
- Stack management
- User tracking limits

User Information classes
User Run

Attaching User Information

- Sometimes users need add an additional information “within” a Geant4 object, e.g. event, track etc.
- This can be done by deriving from a provided Geant4 user information base class and attaching it to the Geant4 object
  - For example you can define MyTrackInformation class derived from G4VUserTrackInformation and attach it to G4Track



MyTrackInformation

- An example of definition of a TrackInformation class
  - With the ID of the primary track which generated this (secondary) track
- Similarly as for hits, using G4Allocator in user information classes helps to improve performance

```
#include "G4VUserTrackInformation.hh"
class TrackInformation : public G4VUserTrackInformation  {
public:
  TrackInformation();
  virtual ~TrackInformation();
  // operators new/delete must be provided when using G4Allocator
  void     SetPrimaryParentID (G4int id) { fPrimaryParentID = id; }
  G4double GetPrimaryParentID() const  { return fPrimaryParentID; }
private:
  G4int  fPrimaryParentID;
};
```

MyTrackInformation (2)

- Typically, a user track information can be created in a user tracking action class and associated to the current track object using a pointer to the Geant4 track manager:

TrackingAction.cc

```
fpTrackingManager
  ->SetUserTrackInformation(new TrackInformation());
```

Trajectory and Trajectory Point

- Trajectory and trajectory point class objects persist until the end of an event.
  - In difference from user information objects associated with track, primary vertex and primary particle which are deleted with their associated Geant4 object
- G4Trajectory and G4TrajectoryPoint are concrete classes provided in Geant4 as defaults.
  - These classes keep only the most common quantities.
  - The are derived from G4VTrajectory and G4VTrajectoryPoint abstract base classes.
- Users can implement their own concrete classes deriving from G4VTrajectory and G4VTrajectoryPoint if they want to keep some additional information
  - User classes should NOT derive from G4Trajectory and G4TrajectoryPoint

Creation of Trajectories

- The creation of a trajectory for the particular track can be switched on/off in the tracking action class
  - Default is off to avoid high memory consumption eg. in high energy EM showers.

```
void TrackingAction::PreUserTrackingAction(const G4Track* aTrack)
{
   // activate storing trajectories
   fpTrackingManager->SetStoreTrajectory(true);
}
```

- User own trajectories can be created in the tracking action class and set to G4TrackingManager:

```
fpTrackingManager->SetTrajectory(new Trajectory());
```

User Run

- Users can also create their own Run class derived from G4Run
- And implement virtual method:

```
void RecordEvent(const G4Event*)
```

  - Here you can get all output of the event so that you can accumulate the quantities accounted in an event to a variable for entire run.
  - This function is automatically invoked by G4RunManager.
- User run class object should be instantiated in GenerateRun() method of the UserRunAction class

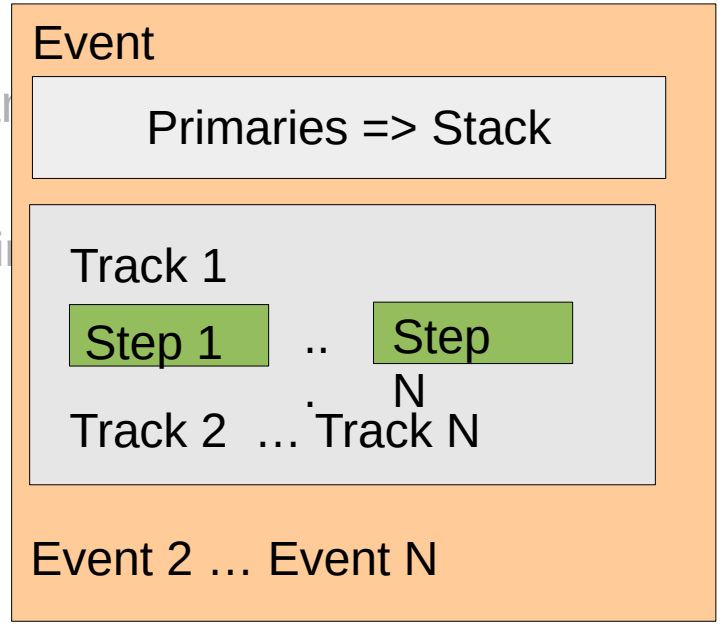
User Information Classes
Overview

- User information classes
  - G4Event - G4VUserEventInformation
  - G4Track - G4VUserTrackInformation
  - G4PrimaryVertex - G4VUserPrimaryVertexInformation
  - G4PrimaryParticle - G4VUserPrimaryParticleInformation
  - G4Region - G4VUserRegionInformation
  - User information class object is deleted when associated Geant4 class object is deleted.
- Base classes to be specialized by users
  - G4Run, G4VHit, G4VDigit, G4VTrajectory, G4VTrajectoryPoint
  - We have already discussed hits in the presentation on scoring

Stack Management

Event in Geant4

- **Event i**s the basic unit of simulation in Geant4.
- At its beginning primary tracks are generated (and pushed onto a stack).
- One ‘track’ at a time is popped from the stack and it is “tracked”
  - Any resulting secondary tracks are pushed back onto the stack.
  - This “tracking” lasts as long as the stack has a track.
- When the stack becomes empty, it’s the end of processing that event
- Classes:
  - An object of G4Event class represents ar
    contains few objects:
    - List of primary vertexes and particles (its i
    - Hits and Trajectory collections (its output)
  - The G4EventManager class coordinates



Track Stacks

- By default, Geant4 has three track stacks.
  - Urgent, Waiting and PostponeToNextEvent
  - The stacks are simple "last-in-first-out" stacks (G4TrackStack)
  - The Urgent stack can optionally use more sophisticated G4SmartTrackStack optimized for performance
  - User can increase the number of stacks
- A Track is popped up only from Urgent stack.
- Once Urgent stack becomes empty, all tracks in Waiting stack are transferred to Urgent stack
- Tracks in PostponeToNextEvent stack are pushed in Urgent stack in the next event

User Stacking Action

- User can change the default stacking behavior in the stacking action class derived from G4VUserStackingAction
- Following three methods have to be implemented:
- G4ClassificationOfNewTrack ClassifyNewTrack(const G4Track*)
  - It decides which stack each newly storing track to be stacked (or to be killed).
  - By default, all tracks go to Urgent stack.
- void NewStage()
  - It is invoked when Urgent stack becomes empty and all tracks in Waiting stack are transferred to Urgent stack
  - All tracks which have been transferred from Waiting stack to Urgent stack can be reclassified by invoking stackManager->ReClassify()
- void PrepareNewEvent()
  - It is invoked at the beginning of each event for resetting the classification scheme.

User Stacking Action

Examples

- basic/B3
  - Stacking action implements killing of all neutrinos generated from Beta decay of Fluor as we are not interested in tracking this neutrino; therefore it is immediately, before being put in a stack.

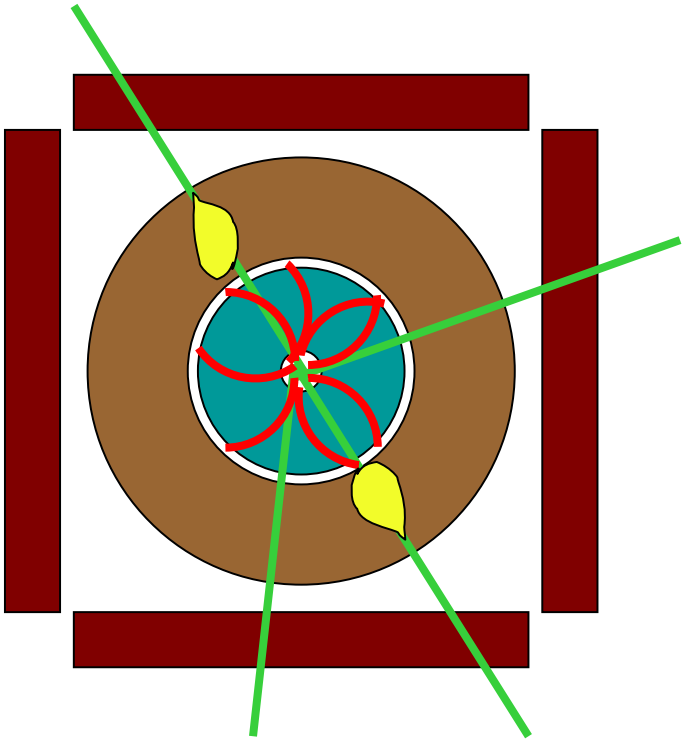
```
G4ClassificationOfNewTrack
StackingAction::ClassifyNewTrack(const G4Track* track)
{
  // keep primary particle
  if (track->GetParentID() == 0) return fUrgent;

  // kill secondary neutrino
  if (track->GetDefinition() == G4NeutrinoE::NeutrinoE())
    return fKill;
  else
    return fUrgent;
}
```

User Stacking Action

Examples

- extended/runAndEvent/RE05
  - A complex example of stacking action
  - Simplified collider detector geometry and event samples of Higgs decays into four muons
    - At the first stage only the primary muons are tracked
      - Check if the primary muons were detected and abort the event if the required condition is not fulfilled
    - At the next stage, only the primary charged particles are tracked only inside the barrel tracking area
      - Check the isolation of muon tracks and abort the event if the condition is not fulfilled
    - At the third stage, all particles in the region of interest are tracked.



Setting Track Status

- The status of a track can be also changed in UserSteppingAction

```
void SteppingAction ::Stepping(const G4Step* step)
{
   G4Track* track = step->GetTrack();
   if ( myCondition) track->SetTrackStatus(fSuspend);
}
```

  - If a track is killed in UserSteppingAction, its physics quantities (energy, charge, etc.) are not conserved but completely lost.

User Tracking Limits

User Tracking Limits

- User limits are artificial limits affecting to the tracking.
- Defined via G4UserLimits class; user can set
  - Maximum allowed step size in a volume
  - Maximum total track length
  - Maximum global track time
  - Minimum remaining kinetic energy (only for charged particles)
  - Minimum remaining range (only for charged particles)
- User limits can be set to logical volume and/or to a region
  - User limits assigned to logical volume do not propagate to daughter volumes.
  - User limits assigned to region propagate to daughter volumes unless daughters belong to another region.
  - If both logical volume and associated region have user limits, those of logical volume win

Limiter Processes

- In addition to instantiating G4UserLimits and setting it to logical volume or region, users have to assign the following process(es) to particle types they want to affect
- G4StepLimiter process
  - Applies **maximum allowed step** size in a volume for the particles which this process is assigned to
  - This process limits a step, but it does not kill a track.
- G4UserSpecialCuts process
  - Applies **remaining limits**:
    - Maximum total track length, maximum global track time, minimum remaining kinetic energy, minimum remaining range (both only for charged particles)
  - This process limits a step and kills the track when the track comes to one of these limits. Step limitation occurs only for the final step.

Limiter Process Builder

- Both G4UserSpecialCuts and G4StepLimiter processes can be added to a modular physics list via G4StepLimiterBuilder
  - This can be done also for Geant4 pre-packaged physics lists
  - An example can be found in basic/B2 example

Summary

- User can attach an information to Geant4 object via user information classes
  - Available for G4Event, G4Track, G4PrimaryVertex, G4PrimaryParticle, G4Region
- Or specialize Geant4 base classes
  - G4Run, G4VHit, G4VDigit, G4VTrajectory, G4VTrajectoryPoint
- User can change the default stacking behavior
  - In the stacking action class derived from G4VUserStackingAction
- G4UserLimits class can be used to apply special limits to tracked particles and let them stop according to user criteria