Stack management, Digitization and Pile-up

Makoto Asai (SLAC)

Geant4 Users Workshop @ SLAC

Feb. 20th, 2002
Introduction

- This tutorial covers the features of
  - Stack management
  - Digitization
  - Multiple events handling and pile-up
- These features are not necessary for naïve simulation. But using them, you can make your simulation program more flexible and efficient.
Stack management

- By default, Geant4 has three track stacks.
  - “Urgent”, “Waiting”, “PostponeToNextEvent”
  - Tracks are popped out only from Urgent stack.
  - Once Urgent stack becomes empty, tracks in Waiting stack are moved to Urgent stack.
  - UserStackingAction class classifies each track which stack to be pushed in (or killed).
  - By utilizing stacks, you can manage priorities of the tracks without “highest priority scan”.
  - You can add stacks for more intelligent stack management.

- Proper stack management and selection/abortion of event gives you better performance of your simulation.
**Stack management**

- **User Stacking Action**
- **Urgent Stack**
- **Waiting Stack**
- **Stacking Manager**
  - Push
  - Pop
- **Postpone To Next Event Stack**
- **Tracking Manager**
  - Security or suspended tracks
  - Process one track
- **Event Manager**
  - Push
  - Pop
  - primary tracks
G4UserStackingAction

- G4UserStackingAction class has three methods.
  - **void NewStage()**
    - Invoked when Urgent stack becomes empty
    - Move tracks in Waiting stack to Urgent stack
    - Or issue `stackManager->ReClassify()` thus all tracks in Waiting stack will be re-examined by `ClassifyNewTrack` method
  - **G4ClassificationOfNewTrack ClassifyNewTrack(const G4Track*)**
    - Invoked every time a track is pushed in
    - Classification
      - fUrgent // pushed into Urgent stack
      - fWaiting // pushed into Waiting stack
      - fPostpone // pushed into PostponeToNextEvent
      - fKill // killed
  - **void PrepareNewEvent()**
    - Invoked at the beginning of new event
ExampleN04 gives you a good example of stack management.

- This example code has a simplified collider detector geometry and event sample of Higgs particle decays into muons.

- It has three stages
  0) Primary muons
  1) Primary charged particles within the tracking region
  2) Primary neutral particles and secondaries in “Region of Interest”
At the beginning of an event (Stage 0), only primary muons are pushed into Urgent stack and others to Waiting stack.

```cpp
G4ClassificationOfNewTrack
ExN04StackingAction::ClassifyNewTrack(const G4Track * aTrack)
{
    G4ClassificationOfNewTrack classification = fWaiting;
    switch(stage)
    {
        case 0:
        {
            if(aTrack->GetParentID()==0)
            {
                G4ParticleDefinition * particleType = aTrack->GetDefinition();
                if((particleType==G4MuonPlus::MuonPlusDefinition())
                   ||(particleType==G4MuonMinus::MuonMinusDefinition()))
                {
                    classification = fUrgent;
                }
            }
        }
        break;
    }
    return classification;
}
When Urgent stack becomes empty (end of stage 0), number of hits in muon counters are examined
   - Event is aborted if reasonable numbers of hits are NOT found, otherwise proceed to next stage.

```cpp
void ExN04StackingAction::NewStage()
{
    stage++;  
    G4int nhits;
    if(stage==1)
    {
        muonHits = (ExN04MuonHitsCollection*)
                    GetCollection("muonCollection");
        nhits = muonHits->entries();
        if(nhits<reqMuon)
        {
            stackManager->clear(); // Event abortion
            return;
        }
    }
    stackManager->ReClassify(); // New classification
    return;
}
```
At stage 1 only primary charged tracks are pushed into Urgent stack

G4ClassificationOfNewTrack
ExN04StackingAction::ClassifyNewTrack(const G4Track * aTrack) {
    G4ClassificationOfNewTrack classification = fWaiting;
    switch(stage) {
    case 1:
        if(aTrack->GetParentID() != 0) { break; }
        if(aTrack->GetTrackStatus() == fSuspend) { break; }
        if(aTrack->GetDefinition()->GetPDGCharge() == 0.) { break; }
        classification = fUrgent;
        break;
    }
    return classification;
}
Each charged primary track except muon is traced in the tracking region and suspended and pushed back to Waiting Stack once it reaches to calorimeter.

```cpp
void ExN04SteppingAction::UserSteppingAction(const G4Step * theStep)
{
    // select charged primary track except muon
    G4Track * theTrack = theStep->GetTrack();
    if(theTrack->GetTrackStatus() != fAlive) { return; }
    if(theTrack->GetParentID() != 0) { return; }
    G4ParticleDefinition * particleType = theTrack->GetDefinition();
    if(((particleType==G4MuonPlus::MuonPlusDefinition())
        || (particleType==G4MuonMinus::MuonMinusDefinition())))
    { return; }
    // check if it is entering to the calorimeter volume
    G4StepPoint * thePrePoint = theStep->GetPreStepPoint();
    G4VPhysicalVolume * thePrePV = thePrePoint->GetPhysicalVolume();
    G4String thePrePVname = thePrePV->GetName();
    if(thePrePVname(0,4) == "calo") { return; }
    G4StepPoint * thePostPoint = theStep->GetPostStepPoint();
    G4VPhysicalVolume * thePostPV = thePostPoint->GetPhysicalVolume();
    G4String thePostPVname = thePostPV->GetName();
    if(thePostPVname(0,4) != "calo") { return; }
    // then suspend the track
    theTrack->SetTrackStatus(fSuspend);
}
```
Remaining scenario

- At the end of stage 1, hits in tracking chambers are examined
  - Event is aborted if reasonable numbers of isolated muons are NOT found
- In stage 2, neutral primary tracks and secondary tracks only inside of “Region of Interest” are pushed into Urgent stack and traced.
  - I.e. Shower is simulated only inside of RoI.
Digitization

- Digit represents a detector output (e.g. ADC/TDC count, trigger signal).
- Digit is created with one or more hits and/or other digits by a concrete implementation derived from G4VDigitizerModule.
- In contradiction to the Hit which is generated at tracking time automatically, the digitize() method of each G4VDigitizerModule must be explicitly invoked by the user’s code (typically at EndOfEventAction).
Digitization

The usages of methods in G4VDigitizerModule is similar to those of G4VSensitiveDetector.

- Constructor
  - Registration of digits collection name(s)

- Digitize() method
  - Get hits from proper hits collections and/or digits from proper digits collections
  - Generate digit(s) and store to dedicated digits collection(s)
  - Set created digits collections to G4DCofThisEvent via StoreDigiCollection() method.
    - G4VDigitizerModule does NOT have initialize() or EndOfEvent() methods.
Usful methods in G4DigiManager

- static G4DigiManager* GetDMpointer();
  - Static method which returns the pointer to G4DigiManager singleton object
- void AddNewModule(G4VDigitizerModule*);
  - Register a digitizer module to G4DigiManager
- G4VDigitizerModule* FindDigitizerModule(G4String);
  - Return the pointer to the digitizer module of given name
- G4int GetHitsCollectionID(G4String);
- G4int GetDigiCollectionID(G4String);
  - Return the hits / digits collection ID of given name
- const G4VHitsCollection* GetHitsCollection(G4int);
- const G4VDigiCollection* GetDigiCollection(G4int);
  - Return the pointer to hits / digits collection of given ID
  - Cast to the concrete collection class
Sample implementation of Digitizer

G4DigiManager* DM = G4DigiManager::GetDMpointer();
G4int HCID = DM->GetHitsCollectionID("/mydet/coll1");

MyDigiCollection* myDigiCol
    = new MyDigiCollection(theDigitizer, collectionName[0]);
MyHitsCollection* myHitsCol
    = (MyHitsCollection*)(DM->GetHitsCollection(HCID));
G4int nHits = myHitsCol->entries();
for(G4int I=0; I<nHits; I++)
{
    MyDigit* aDigit = new MyDigit((*myHitsCol)[i]);
    myDigiCol->insert(aDigit);
}
StoreDigiCollection(myDigiCol);
Multi-event treatment and pile-up

By design, G4EventManager **cannot** process more than one events.

- Previous events kept by G4RunManager are read-only.

Invoke your Digitizer to generate digits / digits collection using collections of previous events. Generated collections are set to the current event.

G4DigiManager provides methods to access to hits / digits collection of previous events.

```cpp
const G4VHitsCollection*
GetHitsCollection(G4int colID,G4int nPrevEv);
const G4VDigiCollection*
GetDigiCollection(G4int colID,G4int nPrevEv);
```

- Don’t forget to invoke `G4RunManager::SetNumberOfEventsToBeStored()` method beforehand.