Fast Simulation

A shortcut to the tracking
I. Introduction:
   – Generalities
   – Parameterisation Features
II. Fast Simulation Components of Geant4:
   – G4VFastSimulationModel
   – Binding concrete models to an envelope
   – G4FastSimulationManagerProcess
   – Summary Picture of Fast Simulation Mechanism
III. Fast Simulation using Ghost Volumes
IV. Example
I. Introduction
Generalities

• Fast Simulation, also called parameterisation, is a shortcut to the tracking.
• Fast Simulation allows you to take over the tracking to implement your own fast physics and detector response.
• The classical use case of fast simulation is the shower parameterisation where the typical several thousand steps per GeV computed by the tracking are replaced by a few ten of deposits per GeV.
• Parameterisations are generally experiment dependent.
Parameterisation features

- Parameterisations take place in an *envelope*. This is typically the mother volume of a sub-system or of a large module of such a sub-system.
- Parameterisations are often *particle type* dependent and/or may apply only to some.
- They are often not applied in complicated regions.
II. Fast Simulation Components of Geant4
G4VFastSimulationModel

- This is the base class allowing to implement concrete parameterisation models.
- It has three pure virtual methods to be overriden:
  - `G4bool IsApplicable(const G4ParticleDefinition *)`
    - Which specify for which particles the model is valid
  - `G4bool ModelTrigger(const G4FastTrack &)`
    - Which allow to decide or not to trigger the model at the current point, in order to avoid to trigger in a « complicated region ».
  - `void DoIt(const G4FastTrack &, G4FastStep &)`
    - Which is the parameterisation properly said, invoked when the model has triggered.
- The G4FastTrack provides input informations to the model (G4Track, envelope informations, …).
- The G4FastStep allows to return back to the tracking the state of the G4Track after parameterisation (alive/killed, position, …) and what are the eventual secondaries created.
Binding concrete models to an envelope

- Concrete models are bound to the envelope through a G4Fast-SimulationManager object.
- This allows several models to be bound to a same envelope.
- The « envelope » is simply a G4LogicalVolume which has received a G4FastSimulationManager.
- All its [grand[[…]]]daughters will be sensitive to the parameterisations.
• The G4FastSimulationManagerProcess is a process providing the \textit{interface} between the tracking and the fast simulation.

• It has to be set to the particles to be parameterised:
  – The process ordering is the following:
    \begin{itemize}
    \item [n-3] ... 
    \item [n-2] Multiple Scattering 
    \item [n-1] G4FastSimulationManagerProcess
    \item [n] G4Transportation
    \end{itemize}
  – It can be set as a discrete process or it must be set as a continuous & discrete process if using ghost volumes (treated later on in this unit).
• The Fast Simulation components are indicated in blue.

• When the G4Track travels inside the volume of the envelope, the G4FSMP looks for a G4FastSimulationManager.

• If one exists, at the beginning of each step in the envelope, the models are messaged to check for a trigger.

• In case a trigger is issued, the model is applied at the point the G4track is.

• Otherwise, the tracking proceeds with a normal step.
III. Fast Simulation using Ghost Volumes
Ghost Volumes (1)

- Ghost volumes allow to define envelopes independently of the volumes of the tracking geometry.
- This allows to group together the electromagnetic and hadronic calorimeters for pion parameterisation for example or to define envelopes for geometries coming out of a CAD system which don’t have a hierarchical structure.
- In addition Ghost volumes are sensitive to particle flavor, allowing to define in a completely independant way envelopes for electrons, envelopes for pion etc…
Ghost Volumes (2)

- Ghost Volumes of a given particle flavor are placed in a clone of the world volume for tracking.
- This is done automatically by a singleton class: the G4GlobalFastSimulationManager.
- The G4FastSimulationManagerProcess provides the additional navigation inside this « parallel » geometry.
- This navigation is done transparently to the user.
- As before, when a parameterisation model attached to a ghost volume issues a trigger, the parameterisation is applied, taking over the tracking.
IV. Example (1)

• Show sample code extracted from example/novice/N05;

• Simulate a (very crude 😞) EM shower:
  – Valid for electrons and gammas;
  – Triggering above 100 MeV;
  – Show in particular a way to collect « hits » created by the parameterisation;
IV.  Example (2)

G4bool ExN05EMShowerModel::IsApplicable(const G4ParticleDefinition& particleType)
{
    return
        &particleType == G4Electron::ElectronDefinition() ||
        &particleType == G4Positron::PositronDefinition() ||
        &particleType == G4Gamma::GammaDefinition();
}

G4bool ExN05EMShowerModel::ModelTrigger(const G4FastTrack& fastTrack)
{
    // Applies the parameterisation above 100 MeV:
    return fastTrack.GetPrimaryTrack()->GetKineticEnergy() > 100*MeV;
}
void ExN05EMShowerModel::DoIt(const G4FastTrack& fastTrack,
                               G4FastStep& fastStep)
{
    G4cout << "ExN05EMShowerModel::DoIt" << G4endl;

    // Kill the parameterised particle:
    fastStep.KillPrimaryTrack();
    fastStep.SetPrimaryTrackPathLength(0.0);
    fastStep.SetTotalEnergyDeposited(fastTrack.GetPrimaryTrack()->
                                      GetKineticEnergy());

    // split into "energy spots" energy according to the shower shape:
    Explode(fastTrack); // Energy spot = (x, y, z, E)

    // and put those energy spots into the crystals:
    BuildDetectorResponse();
}
IV. Example (4)

- To set « energy spot » in sensitive volume, mimic the stepping part regarding hits creation:

```cpp
void ExN05EMShowerModel::AssignSpotAndCallHit(const ExN05EnergySpot &eSpot)
{
    // "converts" the energy spot into the fake G4Step to pass to sensitive detector:
    FillFakeStep(eSpot);
    // call sensitive part: taken/adapted from the stepping:
    // Send G4Step information to Hit/Dig if the volume is sensitive
    G4VPhysicalVolume* pCurrentVolume = 
        fFakeStep->GetPreStepPoint()->GetPhysicalVolume();
    G4VSensitiveDetector* pSensitive;

    if( pCurrentVolume != 0 ) { 
        pSensitive = pCurrentVolume->GetLogicalVolume()->
            GetSensitiveDetector();
        if( pSensitive != 0 ) pSensitive->Hit(fFakeStep);
    }
}
```