Overview of Geant4 Physics

Fermilab Geant4 Tutorial

27-29 October 2003

Dennis Wright (SLAC)

Outline

- Particles and Tracks
- Tracking
- Physics Processes
- Production Cuts
- User Physics Lists

Particles and Tracks (1)

- What is a particle in Geant4?
  - A collection of all the information needed to propagate it through a material
  - Geant4 arranges this information in layers, starting with:

Particle

Simply a definition, no energy, direction, ...

Only one instance of each type

Dynamic Particle

Gives the particle its kinematic properties

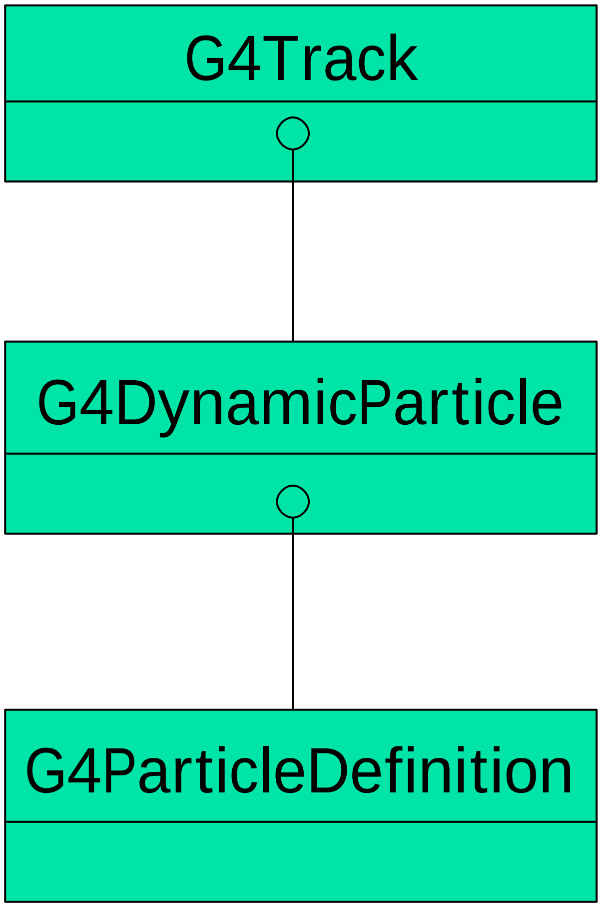
Track

Places the dynamic particle in context

Snapshot of particle

Not a collection of steps

Particles and Tracks (2)



Position, volume, track length
TOF, ID of itself and mother

E, p, polarization, time
pre-assigned decays

PDG info: mass, charge, spin,
lifetime, decay table

Particles and Tracks (3)

- Summing up the previous two slides: a track is a "fully dressed" particle which at any step along its trajectory contains the instantaneous particle information

- Track object lifetime
  - Created by generator or physics process (such as decay of mother)
  - Lives until it
    - decays,
    - goes out of the world volume,
    - goes to zero KE, or
    - is killed by the user

- User access to track info
  - Many public methods: GetPosition(), GetVolume(), GetMaterial(), GetCreatorProcess(), GetMomentum(), GetParticleDefinition(), ...

Particles and Tracks (4)

- Putting particles into your simulation
  - Geant4 kernel takes care of creating tracks, but the user needs to construct all the particle types that will appear in the simulation
  - For example, if you need electrons and protons, the following lines must be included in your code:

    ```
    G4Electron::ElectronDefinition() ;
    G4Proton::ProtonDefinition() ;
    ```

  - Geant4 provides methods which construct entire classes of particles:
    - G4BosonConstructor
    - G4LeptonConstructor
    - G4MesonConstructor
    - G4BaryonConstructor
    - G4IonConstructor
    - G4ShortlivedConstructor

Particles and Tracks (5)

- Particle types available in Geant4 (> 100 by default)
  - quarks, diquarks, gluons
  - photons
  - leptons
  - mesons, baryons
  - nuclei, ions
  - geantinos

- What does Geant4 do with them?
  - Stable, long-lived (> 10^{-14} sec) are tracked
  - K^0 is immediately redefined as K^0_L or K^0_S which is tracked until decay
  - Short-lived never tracked but decayed immediately

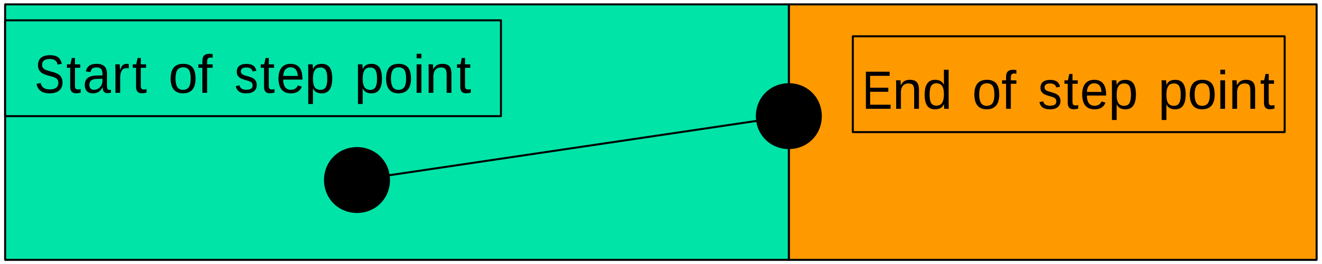
Tracking (1)

- How does Geant4 propagate a particle through a detector ? It must take into account:
  - Track/particle properties
  - All physical processes
  - Volume boundaries
  - Electromagnetic fields

- The job is done by G4SteppingManager, with help from:
  - G4TrackingManager (gets a track from G4EventManager)
  - G4ProcessManager (manages physics processes for each particle type)
  - G4Navigator (locates volume boundaries)
  - G4Transportation (provides a method for integrating the field equation)

Tracking (2)

- The basic element of tracking is the Step
- It consists of two points and the “delta” information of a particle
  - Step length, energy loss during step, change in elapsed time, etc.
- Each point knows which volume it is in
  - If step limited by boundary, end point is located on boundary, but it logically belongs to next volume



Tracking Algorithm (simplified) (1)

- Calculate track velocity

- Each physics process must propose a step length
  - Interaction dependent, look up cross section, calculate MFP
  - "Physical step length" is the minimum of all proposed lengths

- Navigator finds "safety" distance to nearest boundary

- If physical step length is < safety take physical step length

- If not, step is limited by geometry instead of physics
  - Take step to boundary, subtract step length from mean free path of physics processes

Tracking Algorithm (simplified) (2)

- If physics process has limited the step, do the interaction
- Update track properties
- Check for track termination
- If step limited by volume boundary, assign it to next volume
- Invoke G4UserSteppingAction to allow user intervention
- Update processes' MFP

Trajectory

- The Trajectory is a record of a track's history
  - For every step, some information is stored as an object of the G4Trajectory class

- The user can create his own trajectory class by deriving from G4VTrajectory and G4VTrajectoryPoint base classes

- WARNING! Storing trajectories for secondaries generated in a shower may consume large amounts of memory

Physics Processes (1)

- All the work of particle decays and interactions is done by processes
  - Transporation 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 physics of a process may be:
  - Well-located in space → PostStep
  - Not well-located in space → AlongStep
  - Well-located in time → AtRest

Physics Processes (2)

- The most general process may invoke all three of the above actions
  - In that case six methods must be implemented (GetPhysicalInteractionLength() and DoIt() for each action)

- For ease of use, “shortcut” processes are defined which invoke only one.
  - Discrete process (has only PostStep physics)
  - Continuous process (has only AlongStep physics)
  - AtRest process (has only AtRest physics)

Example Processes (1)

- Discrete process: Compton Scattering
  - Step determined by cross section, interaction at end of step (PostStepAction)
- Continuous process: Cerenkov effect
  - Photons created along step, # roughly proportional to step length (AlongStepAction)
- At rest process: positron annihilation at rest
  - No displacement, time is the relevant variable

- These are so-called “pure” processes

Example Processes (2)

- Continuous + discrete: ionization
  - Energy loss is continuous
  - Moller/Bhabha scattering and knock-on electrons are discrete
- Continuous + discrete: bremsstrahlung
  - Energy loss due to soft photons is continuous
  - Hard photon emission is discrete

- In both cases, the production threshold separates the continuous and discrete parts of the process
  - More on this later

- Multiple scattering is also continuous + discrete

Available Processes

- Electromagnetic
  - standard
  - low energy
- Hadronic
  - pure hadronic
  - radioactive decay
  - photo- and electro-nuclear
- Decay
- Optical photon
- Parameterization
- Transportation

Threshold for Secondary Production (1)

- A simulation must impose an energy cut below which secondaries are not produced
  - Avoid infrared divergence
  - Save CPU time used to track low energy particles
- But, such a cut may cause imprecise stopping location and deposition of energy
  - Particle dependence
    - Range of 10 keV γ in Si is a few cm
    - Range of 10 keV e- in Si is a few microns
  - Inhomogeneous materials
    - Pb-scintillator sandwich: if cut OK for Pb, energy deposited in sensitive scintillator may be wrong

Threshold for Secondary Production (2)

- Solution: impose a cut in range
  - Given a single range cut, Geant4 calculates for all materials the corresponding energy at which production of secondaries stops

- During tracking:
  - Particle loses energy by generation of secondaries down to an energy corresponding to the range cut
  - Then the particle is tracked down to zero energy using continuous energy loss. This part is done in a single step.

- The range cut-off represents the accuracy of the stopping position. It does not mean that the track is killed at that energy.

Threshold for Secondary Production (3)

- Geant4 applies the range cut directly to e^-, e^+, γ
  - Geant4 default is 1mm
  - User may change it

- What about protons, muons, pions, etc. ?
  - Proton, e.g., loses energy by emitting δ-rays

  - When it can no longer produce a δ-ray above the energy corresponding to the e- range cut, it is tracked to zero energy by continuous energy loss

Physics Lists (1)

- This is where the user defines all the physics to be used in his simulation

- First step: derive a class (e.g. MyPhysicsList) from the G4VUserPhysicsList base class

- Next, implement the methods:
  - ConstructParticle() - define all necessary particles
  - ConstructProcess() - assign physics processes to each particle
  - SetCuts() - set the range cuts for secondary production

- Register the physics list with the run manager in the main program
  - runManager→SetUserInitialization(new MyPhysicsList);

Physics List (ConstructParticle)

```
void MyPhysicsList::ConstructParticle()
{
    G4Electron::ElectronDefinition();
    G4Positron::PositronDefinition();
    G4Gamma::GammaDefinition();

    G4MuonPlus::MuonPlusDefinition();
    G4MuonMinus::MuonMinusDefinition();
    G4NeutrinoE::NeutrinoEDefinition();
    G4AntiNeutrinoE::AntiNeutrinoEDefinition();
    G4NeutrinoMu::NeutrinoMuDefinition();
    G4AntiNeutrinoMu::AntiNeutrinoMuDefinition();
}
```

Physics List
(SetCuts and ConstructProcess)

```
void MyPhysicsList::SetCuts()
{
    defaultCutValue = 1.0*cm;    //Geant4 recommends 1 mm
    SetCutsWithDefault();
}
```

```
void MyPhysicsList::ConstructProcess()
{
    AddTransportation();            //Provided by Geant4
    ConstructEM();                  //Not provided by Geant4
    ConstructDecay();               //  "      "        "      "
 }
```

Physics List (ConstructEM) (1)

```
void MyPhysicsList::ConstructEM()
{
   theParticleIterator→Reset();
   while( (*theParticleIterator)() ) {
      G4ParticleDefinition* particle = theParticleIterator→Value();
      G4ProcessManager* pm = particle→GetProcessManager();
      G4String particleName = particle→GetParticleName();

      if (particleName == "gamma") {
         pm→AddDiscreteProcess(new G4ComptonScattering);
         pm→AddDiscreteProcess(new G4GammaConversion);
```

PhysicsList (ConstructEM) (2)

```
} else if (particleName == “e-”) {
   pm→AddProcess(new G4MultipleScattering, -1, 1, 1);
   pm→AddProcess(new G4eIonisation,        -1, 2, 2);
   pm→AddProcess(new G4eBremsstrahlung,    -1,-1, 3);
```

These are “compound” processes: both discrete and continuous.

Integers indicate the order in which the process is applied

- first column: process is AtRest
- second column: process is AlongStep
- third column: process is PostStep

Physics List (ConstructDecay)

```
void MyPhysicsList::ConstructDecay()
{
  G4Decay* theDecayProcess = new G4Decay();
  theParticleIterator→reset();
    while( (*theParticleIterator)() ) {
    G4ParticleDefinition* particle = theParticleIterator→value();
    G4ProcessManager* pm = particle→GetProcessManager();
    if (theDecayProcess→IsApplicable(*particle)) {
      pm→AddProcess(theDecayProcess);
    }
  }
}    // Note: there is only one decay process for all particles
```

More Physics Lists

- For a complete EM physics list see novice example N03
  - Best way to start
  - Modify it according to your needs

- Adding hadronic physics is more involved
  - For any one hadronic process, there may be several hadronic models to choose from (unlike EM)
  - Choosing the right models for your application requires care
  - Hadronic physics lists are now provided according to use case

- A physics list for a realistic detector can become cumbersome
  - Consider deriving from G4VModularPhysicsList
  - Has RegisterPhysics method which allows writing “sub” physics lists (muon physics, ion physics, etc.)

Summary

- In Geant4 a track is a snapshot of a particle within the context of a detector. The user decides which particles are useful.
- Geant4 supplies many physics processes which the user must assign to the particles
- Processes and geometry determine where and how a particle interacts
- The precision of particle stopping and the production of secondary particles are determined by a cut in range
- Physics lists are where the user builds particles, processes and sets range cuts