
# Primary particle

Makoto Asai (SLAC Computing Services)

Geant4 Tutorial Course @ Fermi Lab

October 28th, 2003

Geant4

# Primary vertices and particles

- Primary vertices and primary particles should be stored in G4Event before processing an event.
  - G4PrimaryVertex and G4PrimaryParticle classes
    - These classes don't have any dependency to G4ParticleDefinition nor G4Track.
  - Capability of bookkeeping decay chains
    - Primary particles may not necessarily be particles which can be tracked by Geant4.
- Geant4 provides some concrete implementations of G4VPrimaryGenerator.
  - G4HEPEvtInterface
  - G4HEPMCInterface
  - G4GeneralParticleSource
  - G4ParticleGun

# Interfaces to HEPEvt and HepMC

- Concrete implementations of G4VPrimaryGenerator
  - A good example for experiment-specific primary generator implementation
- G4HEPEvtInterface
  - Suitable to /HEPEVT/ common block, which many of (FORTRAN) HEP physics generators are compliant to.
  - ASCII file input
- G4HepMCInterface
  - An interface to HepMC class, which a few new (C++) HEP physics generators are compliant to.
  - ASCII file input or direct linking to a generator through HepMC.

# G4GeneralParticleSource

- A concrete implementation of G4VPrimaryGenerator
- Primary vertex is randomly chosen on the surface of a certain (radioactive) volume.
- Capability of event biasing (variance reduction).
  - By enhancing particle type, distribution of vertex point, energy and/or direction
- Suitable especially to space applications

# G4ParticleGun

- Concrete implementations of G4VPrimaryGenerator
  - A good example for experiment-specific primary generator implementation
- It shoots one primary particle of a certain energy from a certain point at a certain time to a certain direction.
  - Various set methods are available
  - Intercoms commands are also available

# G4VUserPrimaryGeneratorAction

- This class is one of mandatory user action classes to control the generation of primaries.
  - This class itself should NOT generate primaries but invoke **`GeneratePrimaryVertex()`** method of primary generator(s).
  - One of most frequently asked questions is :

    I want “particle shotgun”, “particle machinegun”, etc.
  - Instead of implementing such a fancy weapon, you can
    - Shoot random numbers in arbitrary distribution
    - Use set methods of G4ParticleGun
    - Use G4ParticleGun as many times as you want
    - Use any other primary generators as many times as you want

# G4VUserPrimaryGeneratorAction

- Constructor
  - Instantiate primary generator(s)
  - Set default values to it(them)
- GeneratePrimaries() method
  - Randomize particle-by-particle value(s)
  - Set them to primary generator(s)
  - Invoke **`GeneratePrimaryVertex()`** method of primary generator(s)
  - Never use hard-coded UI commands

# G4VUserPrimaryGeneratorAction

```
void T01PrimaryGeneratorAction::
        GeneratePrimaries(G4Event* anEvent)
{ G4ParticleDefinition* particle;
  G4int i = (int)(5.*G4UniformRand());
  switch(i)
  { case 0: particle = positron; break; ... }
  particleGun->SetParticleDefinition(particle);
  G4double pp =
    momentum+(G4UniformRand()-0.5)*sigmaMomentum;
  G4double mass = particle->GetPDGMass();
  G4double Ekin = sqrt(pp*pp+mass*mass)-mass;
  particleGun->SetParticleEnergy(Ekin);
  G4double angle = (G4UniformRand()-0.5)*sigmaAngle;
  particleGun->SetParticleMomentumDirection
          (G4ThreeVector(sin(angle),0.,cos(angle)));
  particleGun->GeneratePrimaryVertex(anEvent);
}
```

- You can repeat this for generating more than one primary particles.