Primary Particle

Makoto Asai (SLAC)
Geant4 Tutorial Course
Contents

- G4VUserPrimaryGeneratorAction
- Primary vertex and primary particle
- Built-in primary particle generators
  - Particle gun
  - Interfaces to HEPEVT and HEPMC
  - General particle source
- More on particle gun
Primary particle generation
User classes

• Initialization classes
  – Use G4RunManager::SetUserInitialization() to define.
  – Invoked at the initialization
    • G4VUserDetectorConstruction
    • G4VUserPhysicsList

• Action classes
  – Use G4RunManager::SetUserAction() to define.
  – Invoked during an event loop
    • G4VUserPrimaryGeneratorAction
    • G4UserRunAction
    • G4UserEventAction
    • G4UserStackingAction
    • G4UserTrackingAction
    • G4UserSteppingAction

• main()
  – Geant4 does not provide main().

Note: classes written in red are mandatory.
G4VUserPrimaryGeneratorAction

- This class is one of mandatory user classes to control the generation of primaries.
  - This class itself should NOT generate primaries but invoke GeneratePrimaryVertex() method of primary generator(s) to make primaries.
- Constructor
  - Instantiate primary generator(s)
  - Set default values to it(them)
- GeneratePrimaries() method
  - Randomize particle-by-particle value(s)
  - Set these values to primary generator(s)
    - Never use hard-coded UI commands
    - Invoke GeneratePrimaryVertex() method of primary generator(s)
Primary vertex
and primary particle
Primary vertices and particles

- Primary vertices and primary particles are stored in G4Event in advance to 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`.
  - `G4ParticleGun`
  - `G4HEPEvtInterface`, `G4HEPMCInterface`
  - `G4GeneralParticleSource`
Pre-assigned decay products

- Physics generator can assign a decay channel for each individual particle separately, while in Geant4 you cannot specify a decay channel for each particle.
  - Decay chain can be “pre-assigned”.
- A parent particle in the form of G4Track object travels in the detector, bringing “pre-assigned” decay daughters as objects of G4DynamicParticle.
  - When the parent track comes to the decay point, pre-assigned daughters become to secondary tracks, instead of randomly selecting a decay channel defined to the particle type. Decay time of the parent can be pre-assigned as well.
Built-in primary particle generators
• 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 for setting initial values
• One of most frequently asked questions is :
  I want “particle shotgun”, “particle machinegun”, etc.
• Instead of implementing such a fancy weapon, in your implementation of
UserPrimaryGeneratorAction, 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 to make
    overlapping events
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.
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
- G4HepMCIInterface
  - 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
  - Suitable especially to space applications

```cpp
MyPrimaryGeneratorAction::
    MyPrimaryGeneratorAction()
{
    generator = new G4GeneralParticleSource;
}
void MyPrimaryGeneratorAction::
    GeneratePrimaries(G4Event* anEvent)
{
    generator->GeneratePrimaryVertex(anEvent);
}
```

- Detailed description
  http://reat.space.qinetiq.com/gps/
Primary vertex can be randomly chosen on the surface of a certain volume.

- Momentum direction and kinetic energy of the primary particle can also be randomized.
- Distribution could be set by UI commands.

- Capability of event biasing (variance reduction).
  - By enhancing particle type, distribution of vertex point, energy and/or direction.
  - Spherical volume with z biasing, isotropic radiation with theta and phi biasing, integral arbitrary point-wise energy distribution with linear interpolation.
Particle gun
Particle Gun vs. General Particle Source

- **Particle Gun**
  - Simple and naïve
  - Shoot one track at a time
  - Easy to handle.
    - Use set methods to alternate track-by-track or event-by-event values.

- **General Particle Source**
  - Powerful
  - Controlled by UI commands.
    - Almost impossible to control through set methods
  - Capability of shooting particles from a surface of a volume.
  - Capability of randomizing kinetic energy, position and/or direction following a user-specified distribution (histogram).

- If you need to shoot primary particles from a surface of a volume, either outward or inward, GPS is the choice.
- If you need a complicated distribution, not flat or simple Gaussian, GPS is the choice.
- Otherwise, use Particle Gun.
What to do and where to do

• In the constructor of your UserPrimaryGeneratorAction
  – Instantiate G4ParticleGun
  – Set default values by set methods of G4ParticleGun
    • Particle type, kinetic energy, position and direction
• In your macro file or from your interactive terminal session
  – Set values for a run
    • Particle type, kinetic energy, position and direction
• In the GeneratePrimaries() method of your UserPrimaryGeneratorAction
  – Shoot random number(s) and prepare track-by-track or event-by-event values
    • Kinetic energy, position and direction
  – Use set methods of G4ParticleGun to set such values
  – Then invoke GeneratePrimaryVertex() method of G4ParticleGun
  – If you need more than one primary tracks per event, loop over randomization and GeneratePrimaryVertex().

• examples/extended/analysis/A01/src/A01PrimaryGeneratorAction.cc is a good example to start with.
void A01PrimaryGeneratorAction::
    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.