Primary particle generation

Makoto Asai (SLAC)
Geant4 Users Workshop @ SLAC
Feb. 18\textsuperscript{th}, 2002
Primary particles

- Primary vertices and particles should be stored in G4Event before processing the event.
  - G4PrimaryVertex and G4PrimaryParticle classes
    - These classes don’t have dependency to G4ParticleDefinition or G4Track.
  - Capability of bookkeeping decay chains
    - Primary particles are NOT necessarily be particles which can be tracked by Geant4.
- Geant4 provides some concrete implementations of G4VPrimaryGenerator.
  - G4ParticleGun
  - G4HEPEvtInterface
  - G4GeneralParticleSource
G4HEPEvtInterface

- A concrete implementation of G4VPrimaryGenerator
- Suitable to /HEPEVT/ common block, which many of (FORTRAN) HEP physics generators are compliant to.
- ASCII file input
- A good example for experiment-specific primary generator implementation
- Another interface to HepMC class, which a few new (C++) HEP physics generators are compliant to, is planned.
G4GeneralParticleSource

- A concrete implementation of G4VPrimaryGenerator
- Generate radioactive decay fragments
- Primary vertex is randomly chosen within a surface of a certain volume.
  - *spectrum* (defined in terms of energy or momentum)
  - *angular distribution* with respect to a user-defined axis or surface normal
  - *spatial distribution* of particles from 2D or 3D planar surfaces or beam line in Gaussian profile or generated homogeneously within a volume.
- It also provides the option of biasing the sampling distribution. This is advantageous, for example, for sampling the area of a spacecraft where greater sensitivity to radiation effects is expected (e.g. where radiation detectors are located) or increasing the number of high-energy particles simulated, since these may produce greater numbers of secondaries.
### G4GeneralParticleSource

<table>
<thead>
<tr>
<th>2D Surface sources</th>
<th>3D Surface sources</th>
<th>Volume sources</th>
<th>Angular distribution</th>
<th>Energy spectrum</th>
</tr>
</thead>
<tbody>
<tr>
<td>circle</td>
<td>sphere</td>
<td>sphere</td>
<td>isotropic</td>
<td>mono-energetic</td>
</tr>
<tr>
<td>ellipse</td>
<td>ellipsoid</td>
<td>ellipsoid</td>
<td>cosine-law</td>
<td>Gaussian</td>
</tr>
<tr>
<td>square</td>
<td>cylinder</td>
<td>cylinder</td>
<td>user-defined</td>
<td>Linear</td>
</tr>
<tr>
<td>rectangle</td>
<td>paralellapiped</td>
<td>paralellapiped</td>
<td>(through histograms)</td>
<td>Exponential</td>
</tr>
<tr>
<td>Gaussian beam profile</td>
<td>(incl. cube &amp; cuboid)</td>
<td>(incl. cube &amp; cuboid)</td>
<td></td>
<td>power-law</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>bremsstrahlung</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>black-body</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>CR diffuse</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>user-defined</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(through histograms or point-wise data)</td>
</tr>
</tbody>
</table>
G4VUserPrimaryGeneratorAction

★ This class is the mandatory user action class to control the generation of primaries.

★ This class itself should not generate primaries but invoke `GeneratePrimaryVertex()` method of primary generator.

★ 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 method of G4ParticleGun
  
  ★ Invoke G4ParticleGun as many times as you want
G4VUserPrimaryGeneratorAction

- Constructor
  - Instanciate primary generator(s)
  - Set default values to them
- GeneratePrimaries() method
  - Randomize particle-by-particle values
  - Set them to primary generator(s)
  - Invoke GeneratePrimaryVertex() method of primary generator(s)
```cpp
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);
```
G4VUserPrimaryGeneratorAction

    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.