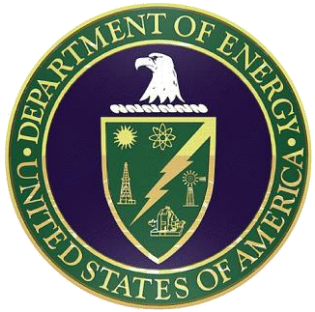


Generating Primary Particles

Geant4 Tutorial, Marshall Space Flight Center
April 2012

Daniel Brandt (based on slides by T. Koi)
based on Geant4 v9.5-p01

Overview

- Mandatory user classes
- Primary generator action class
- Primary generators
  - G4ParticleGun
  - G4GeneralParticleSource
- Decay tables & external decayers

Mandatory User Classes

In order to run a simulation using the Geant4 toolkit the user has to define three mandatory User Classes and the program‘s main-method.

Invoked at initialization using using G4RunManager::SetUserInitialization()

- G4VUserDetectorConstruction – Creates the simulation geometry
- G4VUserPhysicsList – Defines processes to be simulated

Invoked during event loop using using G4RunManager::SetUserAction()

- G4VUserPrimaryGeneratorAction – Creates initial particles

Creating a Primary Generator Action

- The primary generator class is derived from *G4VUserPrimaryGeneratorAction*:

```
class myPrimaryGenerator : public G4VUserPrimaryGeneratorAction
{
  public:
       myPrimaryGeneratorAction();                 //constructor
       virtual ~myPrimarygeneratorAction();        //destructor

       virtual void GeneratePrimaries(G4Event*);
}
```

- At the beginning of every event loop G4RunManager calls *myPrimaryGenerator.GeneratePrimaries(G4Event*)* in order to generate simulation particles.

Generating Primaries

- A Geant4 event begins with a *G4PrimaryVertex* object which holds a number of *G4PrimaryParticle* objects

- *G4PrimaryParticle* and *G4PrimaryVertex* are completely independent of *G4Track* or any particle definitions

- *G4PrimaryParticle* and *G4PrimaryVertex* should not be created directly by the user – instead an instance of a *G4VPrimaryGenerator* object is created and its *GeneratePrimaryVertex(G4Event*)* method is called

G4ParticleGun

- The simplest example of a *G4VPrimaryGenerator* derived class is *G4ParticleGun*

```
#include "G4ParticleGun.hh"

G4ParticleGun particleGun;

myGeneratorAction::GeneratePrimaries(G4Event* anEvent){
  particleGun->SetParticleDefinition(G4Electron::Definition());
  particleGun->SetParticleMomentum(G4ThreeVector(1.0,0,0));
  particleGun->SetParticleEnergy(100.0*keV);

  particleGun->GeneratePrimaryVertex(anEvent);
}
```

- Particle momentum must be a unit vector
- Can use *G4ThreeVector.unit()* to normalize

G4ParticleGun - II

- It is also possible to set polarization and global time for primary particles using *G4ParticleGun:*

```
myGeneratorAction::GeneratePrimaries(G4Event* anEvent){
  particleGun->SetParticleTime(G4double);
  particleGun->SetParticlePolarization(G4ThreeVector);
  ...
}
```

- For a point source, use *G4RandomDirection():*

```
#include "G4RandomDirection.hh"

myGeneratorAction::GeneratePrimaries(G4Event* anEvent){
  particleGun->SetParticleMomentum(G4RandomDirection());
```

Methods provided by G4ParticleGun

•*G4ParticleGun* provides a large number of methods for setting particle attributes

void SetParticleDefinition(G4ParticleDefinition)*
void SetParticleMomentum(G4ParticleMomentum)
void SetParticleMomentumDirection(G4ThreeVector)
void SetParticleEnergy(G4double)
void SetParticleTime(G4double)
void SetParticlePosition(G4ThreeVector)
void SetParticlePolarization(G4ThreeVector)
void SetNumberOfParticles(G4int)

•**NOTE:** To achieve multiple particles with random properties, need to call the relevant *set* methods + *GeneratePrimaryVertex()* multiple times. *SetNumberOfParticles* will create identical particles.

G4GeneralParticleSource

- A sophisticated implementation of G4VPrimaryGenerator

- *G4GeneralParticleSource* should be instantiated and used just like *G4ParticleGun* (it provides *GeneratePrimaries* method)

- *G4GeneralParticleSource* has been created with space applications in mind and can generate particles from point sources, on the surface or throughout the volume of 3D objects

- *G4GeneralParticleSource* provides a number of interactive UI commands. A full Users‘ Manual can be found at http://reat.space.qinetiq.com/gps/

G4GeneralParticleSource - II

•*G4GeneralParticleSource* can be implemented in exactly the same way as *G4ParticleGun*

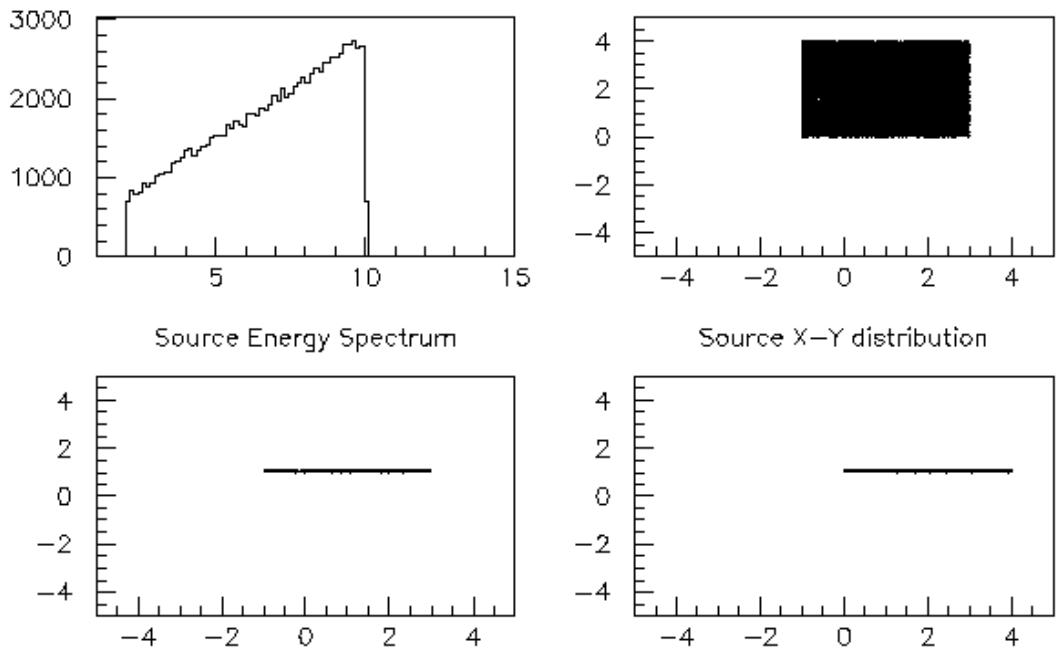
```
//instantiate in constructor
MyPrimaryGeneratorAction::MyPrimaryGeneratorAction()
{ generator = new G4GeneralParticleSource; }

//call GeneratePrimaryVertex from GeneratePrimaries method
void MyPrimaryGeneratorAction::
         GeneratePrimaries(G4Event* anEvent)
{ generator->GeneratePrimaryVertex(anEvent); }
```

•Instead of setting the properties of the particles to be created in the construcor or in *GeneratePrimaries*, the particle properties are set using UI commands

G4GeneralParticleSource - III

- *G4GeneralParticleSource* UI commands can be used from a macro or entered straight into the UI.

```
/gps/particle gamma
/gps/pos/type Plane
/gps/pos/shape Square
/gps/pos/centre 1. 2. 1. cm
/gps/pos/halfx 2. cm
/gps/pos/halfy 2. cm
/gps/ang/type cos
/gps/ene/type Lin
/gps/ene/min 2. MeV
/gps/ene/max 10. MeV
/gps/ene/gradient 1.
/gps/ene/intercept 1.
```



- *Many more examples are available from* [*http://reat.space.qinetiq.com/gps/examples/examples.htm*](http://reat.space.qinetiq.com/gps/examples/examples.htm)

Decay products of primaries

- It is possible to pre-assign the decay chain for primaries

- This is done by using the *SetDecayTable(G4DecayTable*)* method of *G4ParticleDefinition* before calling in *G4UserPhysicsList::ConstrucParticle( );*

- *G4DecayTable* contains a number of *G4DecayChannel* entries, which specify decay modes as well as the relative branching ratio of those modes

Building Decay Tables

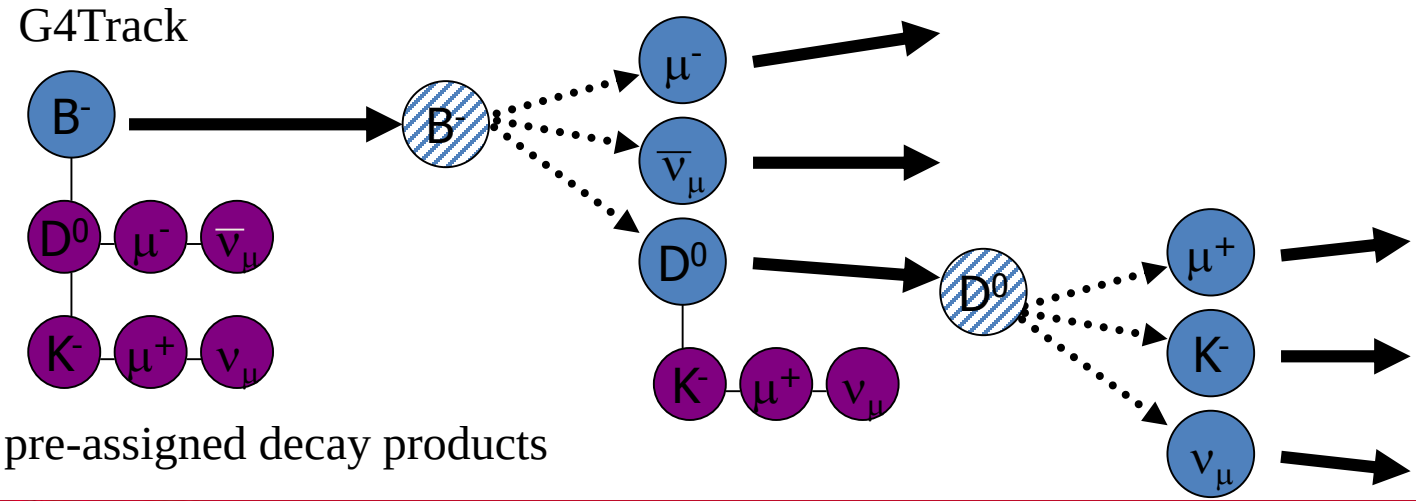
•Example Muonium decay table:

```
//create decay table
G4DecayTable* MuoniumDecayTable = new G4DecayTable();

//Add decay channel to table
MuoniumDecayTable -> Insert(new G4MuonDecayChannel("Mu",1.));

//Add decay table to particle definition
G4Muonium::MuoniumDefinition() -> SetDecayTable(MuoniumDecayTable);
```

•In the example above, *G4MuonDecayChannel* inherits from *G4DecayChannel*, with the decay physics pre-defined and only muon type and branching ratio set in the cosntructor

External Decayer

- In particular, external decayers are used for heavy flavor decays not implemented by Geant4 (c, b, baryons, tau...)
- External physics generators are managed using the *G4VExtDecayer* class
- An example using the *Pythia* external decayer can be found in *$GEANT4SOURCE/examples/extended/eventgenerator/pythia/*



Summary

- *G4VUserPrimaryGeneratorAction* is a mandatory user action class that has to be registered with *G4RunManager*

- Events are generated by *G4VUserPrimaryGeneratorAction::GeneratePrimaries()*

- The user does not create primary vertices directly but uses implementations of *G4VPrimaryGenerator* like *G4ParticleGun* or *G4GeneralParticleSource*

- When the primary particles are short lived, external decayers can be used to control the decay