Generation of a primary event

Luciano Pandola

INFN-LNGS and LNS

Based on a presentation by G.A.P. Cirrone (INFN-LNS)

Outline

- Primary vertex and primary particle
- G4VPrimaryGenerator instantiated via the GeneratePrimaryVertex()
  - The particle gun
  - Interfaces to HEPEVT and HEPMC
  - General Particle Source (or GPS)
- Particle gun or GPS?

User Classes

Initialisation classes

Invoked at the initialization

- G4VUserDetectorConstruction
- G4VUserPhysicsList

Global: only one instance of them exists in memory, shared by all threads (**readonly**). Managed only by the master thread.

Action classes

Invoked during the execution loop

- G4VUserActionInitialization
  - G4VUserPrimaryGeneratorAction
  - G4UserRunAction (*)
  - G4UserEventAction
  - G4UserTrackingAction
  - G4UserStackingAction
  - G4UserSteppingAction

Local: an instance of each action class exists **for each thread**.

(*) Two RunAction's allowed: one for master and one for threads

G4VUserPrimaryGeneratorAction

- It is one of the **mandatory** user classes and it controls the generation of primary particles
  - This class does not directly generate primaries but invokes the **`GeneratePrimaryVertex()`** method of a **generator** to create the initial state
  - It registers the primary particle(s) to the G4Event object
- It has **`GeneratePrimaries(G4Event*)`** method which is purely virtual, so it **must** be implemented in the user class

```
//
// $Id: G4VUserPrimaryGeneratorAction.hh,v 1.5 2006/06/29 21:13:38 gunter Exp $
// GEANT4 tag $Name: geant4-09-03-patch-02 $
//

#ifndef G4VUserPrimaryGeneratorAction_h
#define G4VUserPrimaryGeneratorAction_h 1

class G4Event;

// class description:
//
//  This is the abstract base class of the user's mandatory action class
// for primary vertex/particle generation. This class has only one pure
// virtual method GeneratePrimaries() which is invoked from G4RunManager
// during the event loop.
//  Note that this class is NOT intended for generating primary vertex/particle
// by itself. This class should
//  - have one or more G4VPrimaryGenerator concrete classes such as G4ParticleGun
//  - set/change properties of generator(s)
//  - pass G4Event object so that the generator(s) can generate primaries.
//

class G4VUserPrimaryGeneratorAction
{
  public:
    G4VUserPrimaryGeneratorAction();
    virtual ~G4VUserPrimaryGeneratorAction();

  public:
    virtual void GeneratePrimaries(G4Event* anEvent) = 0;
};

#endif
```

Outline

- Primary vertex and primary particle
- G4VPrimaryGenerator instantiated via the GeneratePrimaryVertex()
  - The particle gun
  - Interfaces to HEPEVT and HEPMC
  - General Particle Source (or GPS)
- Particle gun or GPS?

G4VPrimaryGenerator

- **G4VPrimaryGenerator** is the base class for particle generators, that are called by GeneratePrimaries(G4Event*) to produce an initial state
  - **Notice**: you may have many particles from one vertex, or even many vertices in the initial state
- Derived class from **G4VPrimaryGenerator** must implement the purely virtual method **GeneratePrimaryVertex()**
- Geant4 provides three concrete classes derived by **G4VPrimaryGenerator**
  - G4ParticleGun
  - G4HEPEvtInterface
  - G4GeneralParticleSource

G4ParticleGun

- (Simplest) concrete implementation of **`G4VPrimaryGenerator`**
  - It can be used for experiment-specific primary generator implementation
- It shoots one primary particle of a given energy from a given point at a given time to a given direction
- Various “Set” methods are available (see ../source/event/include/G4ParticleGun.hh)

```
void SetParticleEnergy(G4double aKineticEnergy);
void SetParticleMomentum(G4double aMomentum);
void SetParticlePosition(G4ThreeVector aPosition);
void SetNumberOfParticles(G4int aHistoryNumber);
```

G4VUserPrimaryGeneratorAction: the usual recipe

- Constructor
  - Instantiate primary generator (i.e. **`G4ParticleGun()`**)
    `particleGun = new G4ParticleGun();`
  - (Optional, but advisable): set the default values
    `particleGun -> SetParticleEnergy(1.0*GeV);`
- **`GeneratePrimaries()` mandatory** method
  - Randomize particle-by-particle value, if required
  - Set these values to the primary generator
  - Invoke `GeneratePrimaryVertex()` method of primary generator
    - **`particleGun->GeneratePrimaryVertex()`**

A "real-life" myPrimaryGenerator: constructor & destructor

```
myPrimaryGenerator::myPrimaryGenerator ()
: G4VUserPrimaryGeneratorAction(), fParticleGun(0)
{
  fParticleGun  = new G4ParticleGun();

  // set defaults
  fParticleGun->SetParticleDefinition(
       G4Gamma::Definition());
  fParticleGun->
       SetParticleMomentumDirection(G4ThreeVector(0.,0.,1.));
  fParticleGun->SetParticleEnergy(6.*MeV);
}
```

Instantiate concrete generator

```
myPrimaryGenerator::~myPrimaryGenerator ()
{
  delete fParticleGun;
}
```

Clean it up in the destructor

A "real-life" myPrimaryGenerator: GeneratePrimaries(G4Event*)

```
myPrimaryGenerator::GeneratePrimaries(G4Event* evt)
{
  // Randomize event-per-event
  G4double cosT = -1.0 + G4UniformRand()*2.0;
  G4double phi = G4UniformRand()*twopi;

  G4double sinT = sqrt(1-cosT*cosT);
  G4ThreeVector direction(sinT*sin(phi),sinT*cos(phi),cosT);

  G4double ene = G4UniformRand()*6*MeV;

  fParticleGun->SetParticleDirection(direction);
  fParticleGun->SetParticleEnergy(ene);

  fParticleGun->GeneratePrimaryVertex(evt);
}
```

Sample direction isotropically

Sample energy (flat distr.)

Shoot event

G4ParticleGun

- Commands can be also given interactively by **user interface**
  - But cannot do randomization in this case
- Allows to change primary parameters **between** one run and an other
  - Notice: parameters from the UI could be overwritten in **GeneratePrimaries()**

```
/gun/energy 10 MeV
/gun/particle mu+
/gun/direction 0 0 -1
/run/beamOn 100
/gun/particle mu-
/gun/position 10 10 -100 cm
/run/beamOn 100
```

Change settings

Start first run

Change settings

Start second run

Outline

- Primary vertex and primary particle
- Built-in primary particle generators
  - The particle gun
  - Interfaces to HEPEVT and HEPMC
  - General Particle Source (or GPS)
- Particle gun or GPS?

G4HEPEvtInterface

- Concrete implementation of **`G4VPrimaryGenerator`**
- Almost all event generators in use are written in FORTRAN but Geant4 does not link with any external FORTRAN code
  - Geant4 provides an ASCII file interface for such event generators
- **`G4HEPEvtInterface`** reads an ASCII file produced by an Event generator and reproduce the G4PrimaryParticle objects.
- In particular it reads the **`/HEPEVT/`** fortran block (born at the LEP time) used by almost all event generators
- It generates only the kinematics of the initial state, so does the interaction point must be still set by the user

Outline

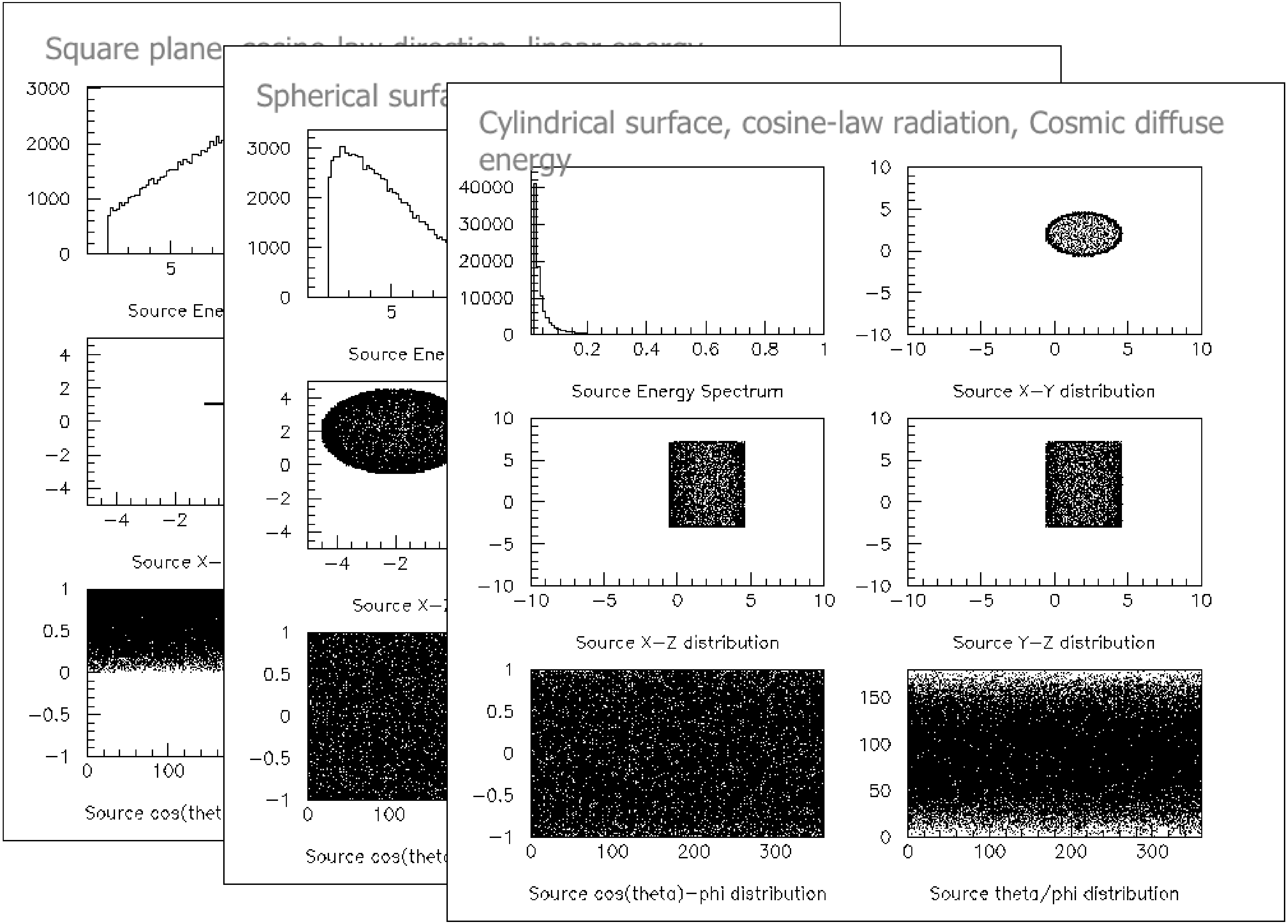
- Primary vertex and primary particle
- Built-in primary particle generators
  - The particle gun
    - Interfaces to HEPEVT and HEPMC
  - General Particle Source (or GPS)
- Particle gun or GPS?

G4GeneralParticleSource()

- **`source/event/include/G4GeneralParticleSource.hh`**
- Concrete implementation of **`G4VPrimaryGenerator`**
  `class G4GeneralParticleSource : public G4VPrimaryGenerator`
- Is designed to replace the G4ParticleGun class
- It is designed to allow specification of multiple particle sources each with independent definition of particle type, position, direction and energy distribution
  - Primary vertex can be randomly chosen on the surface of a certain volume, or within a volume
  - Momentum direction and kinetic energy of the primary particle can also be randomized
- Distribution defined by **UI commands**

G4GeneralParticleSource

- On line manual:
  - Section 2.7 of the Geant4 Application Developer Manual
- /gps main commands
  - **`/gps/pos/type`** (planar, point, etc.)
  - **`/gps/ang/type`** (iso, planar wave, etc.)
  - /**`gps/energy/type`** (monoenergetic, linear, User defined)
  - ....



Square plane
Cylindrical surface, cosine-law radiation, Cosmic diffuse energy
Source Energy Spectrum
Source X−Y distribution
Source X−Z distribution
Source Y−Z distribution
Source cos(theta)−phi distribution
Source theta/phi distribution

GPS documentation

2.7. Geant4 General Particle Source

2.7.1. Introduction

The `G4GeneralParticleSource` (GPS) is part of the Geant4 toolkit for Monte-Carlo, high-energy particle transport. Specifically, it allows the specifications of the spectral, spatial and angular distribution of the primary source particles. An overview of the GPS class structure is presented here. Section 2.7.2 covers the configuration of GPS for a user application, and Section 2.7.3 describes the macro command interface. Section 2.7.4 gives an example input file to guide the first time user.

Spectrum	Abbreviation	Functional Form	User Parameters
mono-energetic	Mono	$I \propto \delta(E-E_0)$	Energy E_0
linear	Lin	$I \propto I_0 + m \times E$	
exponential	Exp	$I \propto \exp(-E/E_0)$	
power-law	Pow	$I \propto E^{\alpha}$	
Gaussian	Gauss	$I = (2\pi\sigma)^{-1/2} \exp[-(E-E_0)^2 / \sigma^2]$	
bremsstrahlung	Brem	$I = \int 2E^2 [h^2c^2 (\exp(-E/kT$ $1)]^{-1}$	
black body	Bbody	$I \propto (kT)^{-1/2} E \exp(-E/kT)$	
cosmic diffuse gamma ray	Cdg	$I \propto [(E/E_b)^{\alpha 1} + (E/E_b)^{\alpha 2}]$	

2.7.3.3. Source position and structure

Command	Arguments	Description and restrictions
/gps/pos/type	dist	Sets the source positional distribution type: *Point* [default], *Plane, Beam, Surface, Volume*.
/gps/pos/shape	shape	Sets the source shape type, after `/gps/pos/type` has been used. For a Plane this can be *Circle, Annulus, Ellipse, Square, Rectangle*. For both Surface or Volume sources this can be *Sphere, Ellipsoid, Cylinder, Para* (parallelpiped).
/gps/pos/centre	X Y Z unit	Sets the centre co-ordinates (X,Y,Z) of the source [default (0,0,0) cm]. The units can only be micron, mm, cm, m or km.
/gps/pos/rot1	$R1_x$ $R1_y$ $R1_z$	Defines the first (x' direction) vector R1 [default (1,0,0)], which does not need to be a unit vector, and is used together with `/gps/pos/rot2` to create the rotation matrix of the shape defined with `/gps/shape`.
/gps/pos/rot2	$R2_x$ $R2_y$ $R2_z$	Defines the second vector R2 in the xy plane [default (0,1,0)], which does not need to be a unit vector, and is used tohgether with `/gps/pos/rot1` to create the rotation matrix of the shape defined with `/gps/shape`.
/gps/pos/halfx	len unit	Sets the half-length in x [default 0 cm] of the source. The units can only be micron, mm, cm, m or km.

ParticleGun vs. GPS

- G4ParticleGun
  - Simple and native
  - Shoots one track at a time
  - Easy to handle
- G4GeneralParticleSource
  - Powerful
  - Controlled by UI commands
    - G4GeneralParticleSourceMessenger.hh
    - Almost impossible to do with the naïve Set methods
  - Capability of shooting particles from a surface or a volume
  - Capability of randomizing kinetic energy, position, direction following a user-specified distribution (histogram)

> •If you need to shot primary particles from a surface of a complicated volume (outward or inward), GPS is the choice
>
> •If you need a complicated distribution, GPS is the choice

When do you need your own derived class of **G4VPrimaryGenerator**

- In some cases, what it provided by Geant4 **does not fit** specific needs: need to write a derived class from **G4VPrimaryGenerator**
  - Must implement the virtual method **GeneratePrimaryVertex(G4Event* evt)**
  - Generate vertices **(G4PrimaryVertex)** and attach particles to each of them **(G4PrimaryParticle)**
  - Add vertices to the event **evt->AddPrimaryVertex()**
- Needed when:
  - You need to interface to a non-HEPEvt **external generator**
    - neutrino interaction, Higgs decay, non-standard interactions
  - Many particles from one vertex, or many vertices
    - double beta decay
  - Time difference between primary tracks

Examples

- **examples/extended/analysis/A01/src/A01PrimaryGeneratorAction.cc** is a good example to start with
- Examples also exist for GPS **examples/extended/eventgenerator/exgps**
- And for HEPEvtInterface **example/extended/runAndEvent/RE01/src/RE01PrimaryGeneratorAction.cc**