初期粒子の発生方法

Geant4 10.3.P3準拠

Geant4　HEP/Space/Medicine 講習会資料

大学共同利用機関法人
高エネルギー加速器研究機構

本資料に関する注意

- 本資料の知的所有権は、高エネルギー加速器研究機構およびGeant4 collaborationが有します
- 以下のすべての条件を満たす場合に限り無料で利用することを許諾します
  - 学校、大学、公的研究機関等における教育および非軍事目的の研究開発のための利用であること
    - Geant4の開発者はいかなる軍事関連目的へのGeant4の利用を拒否します
  - このページを含むすべてのページをオリジナルのまま利用すること
    - 一部を抜き出して配布したり利用してはいけません
  - 誤字や間違いと疑われる点があれば報告する義務を負うこと
- 商業的な目的での利用、出版、電子ファイルの公開は許可なく行えません
- 本資料の最新版は以下からダウンロード可能です
  - http://geant4.kek.jp/lecture/
- 本資料に関する問い合わせ先は以下です
  - Email: lecture-feedback@geant4.kek.jp

目次

1. 初期粒子の発生の基礎
2. ツールキットが提供するParticle Generators
3. ユーザが独自に作るParticle Generator

初期粒子の発生の基礎

初期粒子とは？

- 一つの事象をシミュレーションするにあたり、初期条件として与える粒子を初期粒子（*primary particle*）とよぶ
- primary particleを発生させることは、ユーザが行わなければならない必須のアクション
- primary particleを発生させるには大きく分けて以下の二つの方法がある

① Geant4が提供している*Build-in Primary Particle Generators*を使う

- ユーザにとってもっとも手軽な方法
- 代表的なBuild-in Primary Particle Generators：
  - G4ParticleGun
  - G4GeneralParticleSource （GPS)
  - G4HEPEvtInterface（高エネルギー物理分野用）

② *G4VUserPrimaryGeneratorAction*クラスを継承して、ユーザが独自のprimary particle generatorを実装する

Primary Particle発生の手続き

■ *Build-in Generatorsを利用するにしろ、独自のGeneratorを作るにしろ、primary particleの発生手続きの流れは以下のようになる:*

[注] primary particleの発生手続きは必須ユーザ・アクション

ステップ #1： Primary Particle Generatorの作成

*G4VUserPrimaryGeneratorAction*を継承してユーザ・クラスをつくり、その中にユーザが使いたいprimary particleの発生手続きを書く

ステップ #2 ： 作成したGeneratorをユーザアクションとして登録

上で作ったクラスからGeneratorオブジェクトを作り*G4VUserActionInitialization*に登録する

[注]

- この登録によりGeneratorはG4RunManagerに自動的に渡される
- Geant4 10.0より前のバージョンではG4VUserActionInitializationにGeneratorオブジェクトを登録せずに、直接G4RunMaganerに渡すことができた。しかし、Geant4 10.0以降ではそれは推奨されていない
  （ただし、直接渡す手法を使うことも可能）

Primary Particle発生の手続き: ステップ 1

■ *G4VUserPrimaryGeneratorAction*を継承してユーザ・クラスを作る

```
#include "G4VUserPrimaryGeneratorAction.hh”
class G4Event;

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

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

Primary Particle発生手続きの
ユーザ・クラスを定義

[注]
ユーザ・クラスの名前は任意

ユーザが使いたいPrimary Particle発生手続きを、この関数に記述する – 具体的な手法は後述

[注]
関数内で発生させたprimary particleの情報はG4Eventに付加されてG4RunManagerに渡される

Primary Particle発生の手続き： ステップ ２

■ Primary Particle発生のユーザ・クラスのオブジェクトを *G4VUserActionInitialization*に登録する

G4RunManagerのSetUserInitializationでわたすUserActionInitializationのユーザ•クラスを定義

［注］ primary particleの発生手続きは必須ユーザ・アクションでユーザ・アクションは全て *G4VUserActionInitialization*に登録

ヘッダー・ファイル

```
class UserActionInitialization : public G4VUserActionInitialization
{
  public:
    UserActionInitialization();
    virtual ~UserActionInitialization();

  public:
    virtual void Build() const;
};
```

← この関数内に全てのユーザ・アクションを登録

Build()の実装部分

実装ファイル

```
void UserActionInitialization::Build() const
{
  SetUserAction(new PrimaryGenerator);
}
```

← ユーザ・アクションとして、前のステップ１で定義したPrimaryGeneratorを登録

Primary GeneratorのG4RunManagerへの自動登録

- Primary Generatorの”run manager”への登録は”UserActionInitialization”オブジェクト経由で自動的に行われる ← Primary Generator登録はUser Actionsの一つであ
- Primary Generator登録の流れ

  G4PrimaryGenerator (Primarygenerator) → G4VUserActionInitialization (UserActionInitialization) → G4RunManager

UserActionInitialization.cc

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// UserActionInitialization.cc
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#include "UserActionInitialization.hh"
#include "PrimaryGenerator.hh"

//---------------------------------------------------------------
  UserActionInitialization::UserActionInitialization()
  : G4VUserActionInitialization()
//---------------------------------------------------------------
{}

//---------------------------------------------------------------
  UserActionInitialization::~UserActionInitialization()
//---------------------------------------------------------------
{}

//---------------------------------------------------------------
  void UserActionInitialization::Build() const
//---------------------------------------------------------------
{
  SetUserAction( new PrimaryGenerator() );
}
```

Application_Main.cc

```
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// Geant4 Application: Tutorial course for Hep/Medicine Users
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
#include "Geometry.hh"
#include "UserActionInitialization.hh"

#include "G4RunManager.hh"
#include "G4UIExecutive.hh"
#include "FTFP_BERT.hh"

//---------------------------------------------------------------------
  int main( int argc, char** argv )
//---------------------------------------------------------------------
{
// Construct the default run manager
   G4RunManager* runManager = new G4RunManager;

// Set up mandatory user initialization: Geometry
   runManager->SetUserInitialization( new Geometry );

// Set up mandatory user initialization: Physics-List
   runManager->SetUserInitialization( new FTFP_BERT );

// Set up user initialization: User Actions
   runManager->SetUserInitialization( new UserActionInitialization );

// Initialize G4 kernel
   runManager->Initialize();

// Start interactive session
   G4UIExecutive* uiExec = new G4UIExecutive(argc, argv, "tcsh");
   uiExec->SessionStart();

// Job termination
   delete uiExec;
   delete runManager;

   return 0;
}
```

PrimaryGeneratorの UserActionInitialization への登録

UserActionInitialization のRunManagerへの登録

ツールキットが提供するParticle Generators

G4ParticleGunとは？

- G4ParticleGunでユーザはprimary particleを簡単に発生させることができる
- G4ParticleGunは一度に一つの粒子だけを発生させる
- 発生させるprimary particleに対してユーザが設定可能なパラメータ
  - 粒子の種類／電荷
  - エネルギー／運動量
  - 偏極率
  - 発射位置／発射時間
- 使い方は基本的に以下の二つに分けられる:
  - UIコマンドを用いて発生粒子をコントロールする
  - G4VUserPrimaryGeneratorActionクラスのGeneratePrimaries()の中でG4ParticleGunのメンバー関数を用いて粒子発生をコントロール

[注]

- 上記設定パラメータが全て同じ粒子は、一度に複数個発生可能
- 異なったパラメータの粒子を複数個発生させたければG4ParticleGunを複数用いる（この手法はHands-onで学ぶ）

G4ParticleGunの使い方 ー UIコマンドを用いる

■ 先ずG4VUserPrimaryGeneratorActionを継承したユーザ・クラス（ここではPrimaryGeneratorという名前を使う）のコンストラクタで*G4ParticleGun*を作る

```
PrimaryGenerator::PrimaryGenerator {
  fpParticleGun = new G4ParticeGun();
}
```

[注] fpParticleGunはPrimaryGeneratorクラスのデータメンバーとして定義されている

■ PrimaryGeneratorクラスの関数*GeneratePrimaries(G4Event*)* を以下のように書く:

```
void PrimaryGenerator::GeneratePrimaries(G4Event* anEvent) {
  fpParticleGun-> GeneratePrimaryVertex(anEvent);
}
```

■ 以上の手続きを行うことで、次のようなUIコマンドが使えるようになる

```
/gun/list              使用可能な粒子の種類表示
/gun/particle          発生させる粒子の種類設定
/gun/direction         粒子発生角度（運動量方向ベクトル）設定
/gun/energy            粒子の運動エネルギー設定
/gun/momentum          粒子の運動量設定
/gun/momentumAmp       粒子の運動量絶対値設定
/gun/position          粒子の発生点設定
/gun/time              粒子の発生時間設定
/gun/polarization      粒子の偏極率設定
/gun/number            事象あたりの発生粒子数の設定（同一物理条件の粒子）
/gun/ion               イオン発生の条件設定　[使用方法] /gun/ion Z A Q
```

G4ParticleGunの使い方　ー　提供されている関数を用いる

■ *G4ParticleGun*クラスは以下の関数を提供している

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

■ これらの関数を用いて、UIコマンドでは設定できない事象初期条件を記述することができる：　例えば

- 同一点から発生する複数の粒子（粒子の種類、エネルギー、方向等が異なる）
- 複数の発生点（primary vertex）を持つ事象

[注]　具体的な手法はHands-onで学ぶ

Geant4 General Particle Source (GPS)とは？

- G4ParticleGunが発生する粒子の発生点（primary vertex）は点であったが、GPSでは発生点に広がりを持たすことができる
- GPSの”Source（ソース）”の意味は、発生点が放射線源のように広がりを持つということ
- GPSのソースはユーザが指定する以下のパラメータに従ってprimary particlesを発生させる
  - ソースの形状設定
    - 点、平面、立体、 その他
  - 粒子のエネルギーおよび角度分布設定
    - 一様、ガウス、指数関数、その他 （built-in）
    - ユーザが設定
  - 複数のソース設定 ― 相対強度を設定できる
- GPSは基本的にUIコマンドを用いて使用する

GPSの使い方

- 先ずG4VUserPrimaryGeneratorActionを継承したユーザ・クラス（ここではPrimaryGeneratorという名前を使う）のコンストラクタで*G4GeneralParticleSource*を作る

```
void PrimaryGenerator::PrimaryGenerator {
  gps= new G4GeneralParticleSource();
}
```

［注］ gpsはPrimaryGeneratorクラスのデータメンバーとして定義されている

- 続いて以下のように*GeneratePrimaries(G4Event*)* 関数を記述する：

```
void PrimaryGenerator::GeneratePrimaries(G4Event* anEvent) {
  gps-> GeneratePrimaryVertex(anEvent);
}
```

［注］ 上記の手続きはG4ParticleGunの場合と全く同じ

- 以上で、次のUIコマンドが使えるようになる
  - コマンドおよび使用例は次のGeant4マニュアルを参照

    Geant4 User‘s Guide for Application Developers （2.7節）
  - 使用方法はHands-onで学ぶ

HEPEvtとHepMCへのインターフェース

G4HEPEvtInterface

- HEPEVTはFortranで記述されているHEPの物理過程ジェネレーター(例：pythia)で使われる標準形式
- Fortranのcommon block(/HEPEVT)で表現
- G4HEPEvtInterfaceはGeant4が提供するインターフェイス・クラスで、HEPEVTの情報からprimary particlesを生成させる
- 入力はASCIIファイル

HepMCへのインターフェイス

- HepMCもHEPの物理過程ジェネレーターで使われる標準形式
- HEPEVTを拡張し、C++で記述されている
- Geant4はHepMCとのインターフェスを以下の配布例題コードで提供：

  examples/extended/eventgenerator/HepMC/

ユーザが独自に作るParticle Generator

G4PrimaryVertexとG4PrimaryParticle

■ ユーザが独自のParticle Generatorを使う手順はParticleGun、GPSの場合と全く同じ

- G4VUserPrimaryGeneratorActionを継承したユーザ・クラス（ここではPrimaryGeneratorという名前を使う）の関数*GeneratePrimaries(G4Event*)* を記述する

```
void PrimaryGenerator::GeneratePrimaries(G4Event* anEvent) {
   （ここにユーザ・コードを書く）
}
```

■ ユーザ・コードを書くにあたって、以下の二つのクラスを使用

- G4PrimaryVertex
  primary particle発生点の位置情報を表現するクラス
- G4PrimaryParticle
  primary particleの物理情報（粒子の種類、運動量、など）を表現するクラス

[注]

- これらのクラスはprimary particle発生の記述の時のみに使用
- G4ParticleGunもG4GeneralParticleSourceも内部ではこれらのクラスを使用している
- 実装の具体例はHands-onで学ぶ