Zastosowanie pakietu Geant4 w fizyce jądrowej Wykład 5

Aleksandra Fijałkowska

30 marca 2020

Kilka informacji ogólnych

Wbudowane typy w Geant4:

- G4int
- G4long
- G4float
- G4double
- G4bool
- G4complex
- G4String

Wszystkie typy są zdefiniowane w pliku G4Types.hh

```
typedef double G4double;
typedef float G4float;
typedef int G4int;
typedef bool G4bool;
typedef long G4long;
typedef std::complex<G4double> G4complex;
```

Wyjątkiem jest typ G4String, który jest osobną klasą dziedziczącą po std::string.

Kilka informacji ogólnych cd.

Ponadto Geant4 wykorzystuje szereg klas wbudowanych w bliblotekę CLHEP (Computing Library for High Energy Physics):

- G4ThreeVector (3-elementowy wektor współrzędnych (x,y,z))
- G4RotationMatrix (macierz obrotu 3x3)
- G4LorentzVector (4-elementowy wektor (x, y, z, t))
- G4LorentzRotation (macierz obrotu 4x4)

System jednostek

Geant4 wykorzystuje system jednostek zaczerpnięty z biblioteki HepSystemOfUnits. Podstawowe (domyślne) jednostki to:

- milimetr (mm)
- nanosekunda (ns)
- megaelektronowolt (MeV)
- ładunek pozytonu (eplus)
- kelwin (kelvin)
- mol (mole)
- ładunek pozytonu (eplus)
- kandela (jednostka światłości źródła światła) (candela)
- radian (radian)
- steradian (steradian)

Pozostałe jednostki są zdefiniowane w oparcie o wymienione jednostki podstawowe.
Jeśli użytkownik nie poda jednostki Geant4 zastosuje jednostki domyślne.
Przykład:

```
G4double size = 1*m; //aby rozmiar był zdefiniowany w m
std::cout << size/m << endl; //aby rozmiar został "wypisany" w m
std::cout << energy/keV << " keV";
//wypisz jednostki:
G4UnitDefinition::PrintUnitsTable()
```

Plikiem nagłówkowym załączającym jednostki jest **G4SystemOfUnits.hh**.

Liczby losowe

Za obsługę liczb losowych odpowiada moduł **HEPRandom**, będący kiedyś częścią pakietu Geant4, a obecnie przeniesiony do modułu **CLHEP**.
Moduł **HEPRandom** oferuje zarówno różne silniki, jak i rozkłady.
W praktyce najczęściej używa się funkcji **G4UniformRand()**, definiowaną w pliku **Randomize.hh**:

```
#define G4UniformRand() CLHEP::HepRandom::getTheEngine()->flat()
```

Rozkłady liczb losowych dostarczane przez moduł **HEPRandom**:

- RandFlat - G4MTRandFlat
- RandExponential

```
inline G4double G4MTRandExponential::shoot(CLHEP::HepRandomEngine* anEngine,
                                           G4double mean)
{
  return -std::log(anEngine->flat())*mean;
}
```

- RandGauss

```
inline G4double G4MTRandGauss::shoot(G4double mean, G4double stdDev)
```

- RandPoisson (G4Poisson.hh)

```
inline G4long G4Poisson(G4double mean)
```

Liczby losowe cd.

Geant4 umożliwia zapisanie statusu silnika poprzez wywołanie w skrypcie polecenia:
/random/setSavingFlag true.
Status ten może być wczytany przy kolejnym uruchomieniu symulacji poprzez wywołanie polecenia:
/random/resetEngineFrom currentEvent.rndm.

Primary Generator Action

Każdy projekt osadzony w bibliotece Geant4 wymaga zdefiniowania warunków początkowych Eventu (zdarzenia). Odbywa się to poprzez zaimplementowanie klasy wywodzącej się z klasy abstrakcyjnej **G4VUserPrimaryGeneratorAction** (będziemy ją nazywać **PrimaryGeneratorAction**) Klasa ta posiada czysto wirtualną metodę
virtual void GeneratePrimaries(G4Event*), która musi być zaimplementowana. Jest to miejsce, w którym użytkownik definiuje:

- Typ cząstki początkowej

```
G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
G4ParticleDefinition* particle = particleTable->FindParticle("gamma");
//parametrem metody jest typ G4String - nazwa cząstki
particleGun->SetParticleDefinition(particle);
```

G4ParticleTable jest mapą z wartościami typu G4ParticleDefinition i kluczem G4String. Nie znalazłam metody wypisującej wszystkie dostępne cząstki, można się jednak posiłkować następującymi metodami:

```
G4int size() const
G4ParticleDefinition* GetParticle (G4int index)
const G4String& GetParticleName (G4int index)
```

Primary Generator Action

- Miejsce emisji cząstek

```
particleGun->SetParticlePosition(G4ThreeVector(0.0*cm,0.0*cm,0.0*cm));
```

- Energię cząstek

```
particleGun->SetParticleEnergy(500.0*keV);
```

- Kierunek cząstek

```
particleGun->SetParticleMomentumDirection(G4ThreeVector(1.,0.,0.));
```

Metoda **virtual void GeneratePrimaries(G4Event*)** musi się zakończyć wywołaniem funkcji **GeneratePrimaryVertex(G4Event* event)**.

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

Jedno zdarzenie może mieć więcej niż jedną cząstkę pierwotną! Każde wywołanie funkcji GeneratePrimaryVertex(G4Event* event) powoduje wysłanie nowej cząstki pierwotnej zgodnie z aktualnymi ustawieniami obiektu G4ParticleGun.

G4ParticleGun

Geant4 oferuje dwa rodzaje zmiennej umożliwiające "wysyłanie" cząstek pierwotnych – **G4ParticleGun** oraz **G4GeneralParticleSource**.

W przypadku wykorzystania typu **G4ParticleGun** program wysyła cząstki początkowe z dobrze określoną energią, położeniem i pędem. Inaczej mówiąc obiekt wewnętrznie nie posiada żadnego mechanizmy losowości (co nie oznacza, że tejże losowości nie można zapewnić samemu). Konstruktor klasy przyjmuje liczbę całkowitą, będącą liczbą cząstek, które zostaną wyemitowane z tą samą kinematyką.
Metody klasy **G4ParticleGun**:

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

G4ParticleGun

Tryb interaktywny Geant4 symulacji również umożliwia strowanie cząstkami początkowymi. Do tego służy szereg poleceń "skryptowych"

- /gun/particle particleName
- /gun/direction ex ey z
- /gun/energy energy Unit
- /gun/momentum px py pz Unit
- /gun/position x y z Unit
- /gun/time t Unit
- /gun/polarization Px Py Pz
- /gun/number N
- /gun/ion Z A Q E (liczba atomowa, masa atomowa, ładunek w e, energia wzbudzenia w keV)

Ważne – jeśli chcemy przedefiniowywać parametry cząstek początkowych, nie mogę być one określane "na sztywno" w kodzie w metodzie **GeneratePrimaries(G4Event*)**. Jeśli zależy nam na określeniu jakichś parametrów domyślnych można to zrobić w konstruktorze klasy **PrimaryGeneratorAction**.

G4GeneralParticleSource

Klasa **G4GeneralParticleSource** (GPS) dostarcza szeregu możliwości sterowania emisją cząstek pierwotnych:

- energia cząstek losowana z rozkładu
- losowy kierunek emisji zgodnie z zadanym rozkładem
- rozkład położenia punktu, z którego cząstki są emitowane
- kilka niezależnych źródeł wykorzystanych w obrębie jednej symulacji

GPS jest sterowane z poziomu komend/skryptów. Klasa **G4GeneralParticleSource** bardzo ubogi interfejs publiczny, choć również daje pewne możliwości.

GPS rozkład pozycji

Klasa **G4GeneralParticleSource** (GPS) dostarcza szeregu możliwości sterowania położeniem emisji cząstek pierwotnych:

- punkt
- kształt na płaskiej powierzchni (okrąg, pierścień, elipsa, prostokąt)
- 3 wymiarowy kształt (kula, elipsoida, cylinder, prostopadłościan)
- powierzchnia na 3 wymiarowym kształcie (kula, elipsoida, cylinder, prostopadłościan)

Kilka dostępnych komend (pełna lista znajduje się w rozdziale 2.7.3.3 Geant4 User's Guide for Application Developers):

- /gps/pos/type *Point [default], Plane, Beam, Surface, Volume*
- /gps/pos/shape *możliwe kształty są uzależnione od uprzednio określonego typu*
  - Plane – *Circle, Annulus, Ellipse, Square, Rectangle*
  - Surface oraz Volume – *Sphere, Ellipsoid, Cylinder, Para (prostopadłościan)*
- /gps/pos/centre X Y Z unit

Rozmiary oraz ewentualnie rotacja brył może być określona przy pomocy szeregu kolejnych komend.

GPS rozkład kątowy

Dostępne rozkłady kątowe:

- izotropowy w 4π
- izotropowy w 2π (płaski)
- wiązka jednowymiarowa
- wiązka dwuwymiarowa
- zdefiniowany przez użytkownika

/gps/ang/type – *iso [default], cos, planar, beam1d, beam2d, focused, user*

Możliwe jest określenie górnej i dolnej granicy kątów θ i ϕ.
Kierunki osi określa świat (supermatka).

W przeciwieństwie do powszechnie przyjmowanej konwencji kierunek emisji P_x, P_y i P_z jest wyznaczany następująco:
$P_x = -sin\theta cos\phi$
$P_y = -sin\theta sin\phi$
$P_z = -cos\theta$

GPS rozkład kątowy

Rozkład zdefiniowany przez użytkownika polega na określeniu rozkładów kąta θ i ϕ w formie histogramów.
W tym celu należy wykorzystać komendę
/gps/hist/type
z parametrem *theta* lub *phi*.
Punkty histogramu wczytuje się pojedynczo wykorzystując komendę
/gps/hist/point
przyjmującą dwa argumenty – górna granica binu, wartość
Histogram można też wczytać z pliku
/gps/hist/file *HistFile*
Plik musi zawierać dwie kolumny – górną granicę binu oraz jego wartość.
Granice binów powinny być wyznaczone w domyślnych jednostkach Geant4 (np MeV). Wyjątek stanowi pierwsza linia, która powinna zawierać tylko dolną granice binu.
Dopuszczalny maksymalny rozmiar histogramu wynosi 1024 biny.

GPS rozkład energii

Dostępne rozkłady energii:

Typ	Skrót	Wyrażenie	Parametry
Monoenergtyczny	Mono	$I \sim \delta(E - E_0)$	energia E_0
Liniowy	Lin	$I \sim I_0 + m \cdot E$	I_0, m
Eksponencjalny	Exp	$I \sim exp(\frac{-E}{E_0})$	E_0
Potęgowy	Pow	$I \sim E^{\alpha}$	α
Gaussowski	Gauss	$I \sim \frac{1}{\sqrt{2\pi\sigma}} exp(\frac{-(E-E_0)}{\sharp})$ (???)	E_0, σ

Dokumentacja podaje jeszcze kilka rozkładów, ale po przykładzie Gaussowskim straciłam do nich zaufanie.
Tabela 2.7.3.5 w Geant4 User's Guide for Application Developers prezentuje polecenia skryptowe sterujące rozkładem energii GPS.

Zadanie na dziś

Do poniższych zadań wykorzystaj obiekt typu **G4ParticleGun**

- Napisz metodę

```
G4ThreeVector GenerateIsotropicDirection(G4double thetaMin,
                                         G4double thetaMax,
                                         G4double phiMin,
                                         G4double phiMax)
```

  zwracającą wektor pędu wylosowanego z rozkładu izotropowego w granicach thetaMin- thetaMax oraz phiMin - phiMax.
  Wersja łatwiejsza – rozkład może obejmować cały kąt 4π, bez określania górnej i dolnej granicy.
  Wskazówka: Skorzystaj z metody **G4UniformRand()** wymagającej nagłówek **Randomize.hh**, która zwraca liczbę losową od 0 do 1.
  Pytanie: Jakie warunki musi spełniać rozkład izotropowy?

- Zaimplementuj metodę

```
void GeneratePositionIncident(G4Event* anEvent)
```

  zapoczątkowujący Event wysłaniem jednego pozytonu o energii 600 keV izotropowo ze środka geometrii.
  Wywołaj metodę wewnątrz funkcji void **GeneratePrimaries(G4Event* anEvent)**.

Zadanie na dziś cd.

- Napisz metodę

```
void GenerateBackgroundIncident(G4Event* anEvent)
```

zaczynający Event emisją kwantu γ izotropowo z losowego położenia wewnątrz świata (obiekt World).
Dla uproszczenia załóżmy, że promieniowanie γ pochodzi z czterech izotopów promieniotwórczych występujących naturalnie:

- ^{40}K (1461 100%)
- ^{208}Tl (511 23%; 583 85%; 861 13%; 2615 100%)
- ^{214}Pb (295 18%; 352 36%)

oraz, że jąder ^{40}K jest 2 razy więcej niż ^{208}Tl i ^{214}Pb.