Zastosowanie pakietu Geant4 w fizyce jądrowej Wykład 5

Aleksandra Fijałkowska

15 listopada 2018

Ogłoszenia

Przypominam, że zajęcia w dniach 23 i 29 listopada NIE ODBĘDĄ się.

Obligatoryjne moduły symulacji

- geometria (klasa **DetectorConstruction**)
- procesy fizyczne (klasa **PhysicsList**, będzie omawiana na kolejnych zajęciach)
- kinemetyka cząstek pierwotnych (klasa **PrimaryGeneratorAction**)

W oparciu o dostarczone dane Geant4 wykonuje symulacje. Wynikiem symulacji może być np. widmo energii zdeponowanej w detektorze, rozkład ładunku, dawka, liczba wyemitowanych fotonów optycznych itp.
Dostęp do tych informacji wymaga uzupełniania podstawowej części kodu o dodatkowe elementy.

Wyniki symulacji, **G4Step**

Istnieją trzy sposoby aby uzyskać dostęp do wyników symulacji

- Korzystając z klasy **SteppingAction**, implementującej klasę bazową **G4UserSteppingAction**. Klasa ta posiada wirtualną metodę **UserSteppingAction(const G4Step*)**, która jest wykonywana po każdym kroku (**Step**).
  Obiekt typu **G4Step** :

```
G4Track *  GetTrack () const
G4double  GetStepLength () const
G4double  GetTotalEnergyDeposit () const
G4double  GetNonIonizingEnergyDeposit () const
G4ThreeVector  GetDeltaPosition () const
G4double  GetDeltaTime () const
G4ThreeVector  GetDeltaMomentum () const
G4double  GetDeltaEnergy () const
```

G4Track :

```
const G4ParticleDefinition *  GetParticleDefinition () const
const G4ThreeVector &  GetPosition () const
G4VPhysicalVolume *  GetVolume () const
G4Material *  GetMaterial () const
G4Material *  GetNextMaterial () const
G4double  GetKineticEnergy () const
G4double  GetTotalEnergy () const
const G4ThreeVector &  GetMomentumDirection () const
G4ThreeVector  GetMomentum () const
```

Wyniki symulacji, **G4VSensitiveDetector**

- Implementując klasę **G4VSensitiveDetector** oraz czysto wirtualną metodę **G4bool ProcessHits(G4Step*aStep,G4TouchableHistory*ROhist)**.
  Warto zwrócić uwagę, że metoda **ProcessHits** również wykorzystuje fakt dostępu do obiektu **G4Step** oraz wszystkich jego publicznych metod, więc od strony implementacji metody **UserSteppingAction(const G4Step*)** i **G4bool ProcessHits(G4Step*aStep,G4TouchableHistory*ROhist)** są dość podobne.
  Interesujące dane otrzymane z kroku są zapisywane w klasie dziedziczącej po **G4VHit**, zastosowanie tej metody wymaga zaimplementowana zarówno **G4VSensitiveDetector**, jak i **G4VHit**.
  Zwyczajowo tworzy się dwie klady o zbliżonej nazwie, np. **NaISD** i **NaIHit**.
  Dostęp do hitów (umieszczonych jako zbiór w **G4THitsCollection**) uzyskuje się w klasie implementującej **G4UserEventAction** (w naszym projekcie nazywa się **EventAction**, w metodzie **void EndOfEventAction(const G4Event*)**.
  Wymogiem wykorzystania **G4THitsCollection** jest zadeklarowanie **G4Allocators** dla implementacji klasy **G4VHit**, a także operatora **new()** i **delete()**.

Wyniki symulacji, **G4VPrimitiveScorer**

- Korzystając z gotowej, wbudowanej w Geant4, gotowej implementacji klasy **G4VSensitiveDetector** czyli tzw. **G4MultiFunctionalDetector**. Klasa **G4MultiFunctionalDetector** umożliwia rejestrację obiektów **G4VPrimitiveScorer**, czyli prostych liczników (wymieniam tylko wbrane):
  - **G4PSTrackLength** – zwraca długość toru, liczoną jako sumę długości kroków, jakie pokonała cząstka wewnątrz detektora
  - **G4PSEnergyDeposit** — zwraca całkowitą energię zdeponowaną w detektorze
  - **G4PSDoseDeposit** — zwraca całkowitą energię zdeponowaną w detektorze podzieloną przez masę detektora. Masa jest wyznaczana w oparciu o gęstość i objętość określone w **G4VSolid** oraz **G4LogicalVolume**
  - Grupa liczników zwracająca prąd i strumień cząstek przelatujących przez powierzchnię detektora
  - **G4PSMinKinEAtGeneration** – zwraca minimalną energię kinetyczną cząstki wtórnej wytworzonej w detektorze (tu nie ma sumowania, podawana jest wyłącznie najmniejsza wartość energii)
  - **G4PSNofSecondary** – zwraca liczbę cząstek wtórnych, wygenerowanych w detektorze
  - **G4PSNofStep** – zwraca liczbę kroków w detektorze
  - **G4PSCellCharge** – zwraca całkowity ładunek cząstek zatrzymanych w detektorze

Pełną listę liczników zawiera rozdział 4.4.5. Geant4 User's Guide for Application Developers.

Wyniki symulacji, **G4VPrimitiveScorer**

Wyniki wyznaczone z prostych liczników są zapisywane w **G4THitsMap**, do którego użytkownik ma dostęp w metodzie **void EndOfEventAction(const G4Event*)** klasy **EventAction**. Mapa jest indeksowana liczbą naturalną będącą numerem kopii obszaru logicznego (lub jego przodka).
Proste liczniki sumują interesujące wielkości w każdym kroku, a więc oferują informację „zbiorczą" dla całego zdarzenia (Eventu).

Na dzisiejszych i kolejnych zajęciach będziemy implementować wszystkie te metody.

Zadanie 1, **G4Step**

Znajdź całkowitą energię zdeponowaną w kręgosłupie fantomu.
Założenia:

- Ze środka fantomu (całej geometrii) emitowany jest pozyton o energii 587 keV (średnia energia pozytonu emitowanego w przemianie β ^{89}Sr) z izotropowym rozkładem kątowym.
- Jako wynik chcemy uzyskać listę depozytów energii w danym zdarzeniu w postaci pliku tekstowego zawierającego kolumnę z numerem zdarzenia oraz odpowiadającym mu całkowitym depozytem energii.
- W klasie **SteppingAction** stwórz zmienną, która będzie trzymała depozyt energii w kręgosłupie (np. **spineEnergyDep**).
- W każdym kroku zmienna **spineEnergyDep** musi być zwiększana o depozyt energii w tym kroku pod warunkiem, że krok miał miejsce w kręgosłupie. Algorytm ten należy napisać wewnątrz metody **void SteppingAction::UserSteppingAction(const G4Step* theStep)**.
- Depozyt energii można uzyskać dzięki metodzie klasy **G4Step**, **GetTotalEnergyDeposit()** (proszę nie zapominać o jednostkach!)

Zadanie 1, **G4Step**

Znajdź całkowitą energię zdeponowaną w kręgosłupie fantomu.
Założenia cd.:

- Informacje o objętości (zarówno G4VPhysicalVolume, jak i G4LogicalVolume), w której zaszedł krok, można uzyskać z następujących metod:

```
G4StepPoint* preStepPoint = theStep->GetPreStepPoint();
G4VPhysicalVolume* preStepPV = preStepPoint->GetPhysicalVolume();
G4LogicalVolume* preStepLV = preStepPV->GetLogicalVolume();
```

- Zarówno sczytywanie danych, jak i czyszczenie zmiennej **spineEnergyDep** musi się odbyć w metodzie **void EventAction::EndOfEventAction(const G4Event* anEvent)**

Zadanie 2, **G4VPrimitiveScorer**

Znajdź całkowitą energię zdeponowaną kolejnych kryształach NaI. Skorzystaj z prostego licznika **G4PSEnergyDeposit**.
Cząstkę pierwotną wygeneruj zgodnie z poprzednim zadaniem.
Kroki:

- Utworzyć obiekt klasy **G4MultiFunctionalDetector** (nazwa jest ważna, jednoznacznie identyfikuje detektor i umożliwia dostęp do danych)

```
G4MultiFunctionalDetector* detector =
new G4MultiFunctionalDetector("naISensitiveDet");
```

- Zarejestrować go do Sensitive Detector Manager-a (**G4SDManager**)

```
G4SDManager::GetSDMpointer()->AddNewDetector(detector);
```

lub w dwóch krokach:

```
G4SDManager* SDmanager = G4SDManager::GetSDMpointer();
SDmanager->AddNewDetector(detector);
```

- Przypisać MultiFunctionalDetector do interesujących obszarów logicznych

```
naILog->SetSensitiveDetector(detector);
```

- Utworzyć obiekt klasy **G4VPrimitiveScorer**, konstruktor przyjmuje nazwę oraz liczbę całkowitą, oznaczającą rząd przodka, określającego numer kopii

```
G4VPrimitiveScorer* energyDepScorer = new G4PSEnergyDeposit("eDep",depth);
```

- Zarejestrować PrimitiveScorer do MultiFunctionalDetector

```
detector->RegisterPrimitive(energyDepScorer);
```

Zadanie 2, **G4VPrimitiveScorer** cd.

- Każdy **G4VPrimitiveScorer** generuje mapę **G4THitsMap<G4double>**. Indeksem mapy jest numer kopii obszaru logicznego do którego przypisany jest MultiFunctionalDetector (lub numer kopii jego przodka, w zależności od parametru, który wpisaliśmy w konstruktorze klasy **G4VPrimitiveScorer**.
  Wartościami w mapie są wielkości, które miał zliczać **G4VPrimitiveScorer** (w naszym przypadku depozyty energii).
- Dostęp do **G4THitsMap<G4double>** można uzyskać po skończonym zdarzeniu (Event). Są one zalokowane w obiekcie typu **G4HCofThisEvent**, do którego dostęp uzyskuje się ze zmiennej **G4Event**. W tym celu należy:
  1. Znaleźć unikalny numer interesującej mapy. Numery te przetrzymuje G4SDManager (sensitive detector manager), który jest singletonem. Dostęp do niego można uzyskać dzięki publicznej, statycznej metodzie **GetSDMpointer()**. Obiekt ten ma publiczną metodę **G4int GetCollectionID (G4String colName)** zwracającą numer **G4THitsMap<G4double>** (kolekcji).
     Argumentem wejściowym jest nazwa kolekcji, składająca się z nazwy **G4MultiFunctionalDetector** / nazwy **G4VPrimitiveScorer**. W naszym przykładzie byłoby to „nalSensitiveDet/eDep”

Zadanie 2, **G4VPrimitiveScorer** cd.

2. Wyciągnąć z Eventu (po jego zakończeniu) obiekt typu G4HCofThisEvent (hits collections of an event), odpowiada za to metoda **GetHCofThisEvent()** klasy **G4Event**. Może się zdarzyć, że metoda zwróci pusty wskaźnik (np. jeśli w zdarzeniu nie było żadnego depozytu energii), przed przejściem dalej należy to sprawdzić.
3. Metoda **G4VHitsCollection* GetHC (G4int i)** klasy **GetHCofThisEvent()** zwraca kolekcję odpowiadającą zadanemu numerowi **i**. Typ **G4VHitsCollection** jest typem bazowym dla **G4THitsMap**<**T**> oraz G4THitsCollection < T > (którym zajmiemy się na przyszłych zajęciach), aby otrzymać typ G4THitsMap < G4double > * należy wykonać na niego rzutowanie (**dynamic_ cast** < **G4THitsMap** < **G4double** > * >).

Zadanie 2, **G4VPrimitiveScorer** cd.

4. Po długich bojach mamy już obiekt **G4THitsMap<G4double>** zawierający interesujące nas wyniki symulacji. Aby wyciągnąć te dane można skorzystać z operatora **[] (T* operator[] (G4int key) const)**. Proszę zwrócić uwagę na fakt, że operator przyjmuje klucz z mapy. Może się okazać, że mapa posiada tylko klucz równy np 4 (tylko 4 kopia detektora ma niezerowy depozyt energii). Nie można z góry zakładać, że klucz 0, 1, 2 itp istnieje. W przykładach, ktre widziałam użytkownicy zazwyczaj sprawdzają, czy mapa jest niezerowa dla kolejnych możliwych kluczy (z góry wiedząc ile może być kopii detektora). Alternatywnie można skorzystać z metody **std::map < G4int, T * > * GetMap ()const**, która zwraca typ **std::map**, na którym można już operować jak na każdej przyzwoitej mapie.
   Drugą trudnością jest fakt, że metoda **GetHC** zwraca wskaźnik do mapy, a operator **[]** wskaźnik do wartości, żeby uzyskać G4double musimy zrobić podwójne odwskaźnikowanie:
   G4double depozytEnergii =*((*mapaDepozytów)[klucz]);

Zadanie 3

Porównaj wyniki otrzymywane z prostego licznika z danymi otrzymywanymi z kroku.