Zastosowanie pakietu Geant4 w fizyce jądrowej Wykład 9

Aleksandra Fijałkowska

1 maja 2020

Wyniki symulacji, Nasz licznik

Na poprzednich zajęciach omówiliśmy **Primitive Scorers**, proste liczniki, czyli obiekty zaliczające wybrane wielkości (np. depozyt energii).Mogliśmy do jednej obiętości przypisać więcej niż jeden licznik, ale działają one niezależnie. Często chcemy jednak zbadać bardziej skomplikowane cechy detektora i symulacji (np chcemy jednocześnie depozyt energii, czas oddziaływania, typ cząstki itp). W takiej sytuacji musimy stworzyć kawałek kodu, który po pierwsze wydobędzie te dane a po drugie umieści je w jakimś rozsądnym obiekcie. Przypominam, że wyniki działania prymitywnych liczników były zwykłymi liczbami zmiennoprzecinkowymi typu **double**. W przypadku „naszych liczników" możemy stworzyć strukturę, która będzie trzymała wszystko to, co jest nam potrzebne.

Będziemy potrzebować dwóch typów klas:

- klasę wywiedzioną z klasy bazowej **G4VSensitiveDetector**, zawierającą implementację czysto wirtualnej metody **G4bool ProcessHits(G4Step*aStep,G4TouchableHistory*ROhist)**, w której wyciągamy interesujące wyniki symulacji.
- klasę wywiedzioną z klasy **G4VHit**, w której będziemy trzymać wyniki symulacji

G4VSensitiveDetector

Implementując klasę **G4VSensitiveDetector** oraz czysto wirtualną metodę **G4bool ProcessHits(G4Step*aStep,G4TouchableHistory*ROhist)**, warto zauważyć, 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()**.

Sposób implementacji

Fragment pliku nagłówkowego klasy G4VSensitiveDetector:
class description:
This is the abstract base class of the sensitive detector. The user's sensitive detector which generates hits must be derived from this class. In the derived class constructor, name(s) of hits collection(s) which are made by the sensitive detector must be set to „collectionNameśtring vector.

```
class G4VSensitiveDetector
{
  public:
      G4VSensitiveDetector(G4String name);
      G4VSensitiveDetector(const G4VSensitiveDetector &right);
      // Constructors. The user's concrete class must use one of these
      // constructors by the constructor initializer of the derived class.
      // The name of the sensitive detector must be unique.

      virtual ~G4VSensitiveDetector();
      virtual void Initialize(G4HCofThisEvent*);
      virtual void EndOfEvent(G4HCofThisEvent*);
      //  These two methods are invoked at the begining and at the end of each
      // event. The hits collection(s) created by this sensitive detector must
      // be set to the G4HCofThisEvent object at one of these two methods.
```

Sposób implementacji

Fragment pliku nagłówkowego klasy G4VSensitiveDetector, cd.:

```
  protected:
      virtual G4bool ProcessHits(G4Step*aStep,G4TouchableHistory*ROhist) = 0;
      //  The user MUST implement this method for generating hit(s) using the
      // information of G4Step object. Note that the volume and the position
      // information is kept in PreStepPoint of G4Step.

      virtual G4int GetCollectionID(G4int i);
      //  This is a utility method which returns the hits collection ID of the
      // "i"-th collection. "i" is the order (starting with zero) of
      //the collection whose name is stored to the collectionName
      // protected vector.

      G4CollectionNameVector collectionName;
      //  This protected name vector must be filled at the constructor of
      //the user's concrete class for registering the name(s) of hits
      // collection(s) being created by this particular sensitive detector.
...
#endif
```

Sposób implementacji, G4VSensitiveDetector

Klasa Sensitive Detector w praktyce (kroki)

- Stworzyć klasę wywiedzioną z **G4VSensitiveDetector**
- Wywołać konstruktor klasy matki czyli **G4VSensitiveDetector**
- W kostruktorze klasy stworzyć nazwę „naszej" HitsCollection i dodać ją do wektora **collectionName**
- W metodzie **Initialize()** utworzyć obiekt **HitCollection** i dodać ją do obiektu **G4HCOfThisEvent**
- Zaimplementować metodę **ProcessHits()** – w niej tworzymy „hity" i dodajemy je do kolekcji

G4VHit – worek na dane

Następnie należy stworzyć obiekt, który będzie trzymał interesujące dane.
Klasą bazową jest **G4VHit**.
Geant4 wymusza, aby do obiektu **G4VHit** napisać także operator przypisania i konstruktor kopiujący.

```
class NaIHit : public G4VHit
{
  public:
    //przykładowe dane, które możemy wkładać do konstruktora
    NaIHit(G4int moduleIndexVal,
           G4double energyDepVal, G4double hitTimeVal);
    virtual ~NaIHit();
    NaIHit(const NaIHit &right);
    const NaIHit& operator=(const NaIHit &right);
    G4int operator==(const NaIHit &right) const;

    inline void *operator new(size_t);
    inline void operator delete(void *aHit);

    //tu powinien być zbiór Getterów i Setterów
  private:
    G4int moduleIndex;
    G4double energyDepVal;
    G4double hitTimeVal;
};
```

G4VHit – ostatni element

Plik nagłówkowy cd. (po skończonej deklaracji klasy, ale w pliku *.hh)

```
//WAZNE i OBOWIĄZKOWE, zawsze wyglada tak samo
typedef G4THitsCollection<NaIHit> NaIHitsCollection;

extern G4ThreadLocal G4Allocator<NaIHit>* NaIHitAllocator;

inline void* NaIHit::operator new(size_t){
  if(!NaIHitAllocator)
      NaIHitAllocator = new G4Allocator<NaIHit>;
  return (void *) NaIHitAllocator->MallocSingle();
}

inline void NaIHit::operator delete(void *aHit){
  NaIHitAllocator->FreeSingle((NaIHit*) aHit);
}
```

G4VHit – ostatni element

Plik źródłowy

```
//Stały punkt
G4ThreadLocal G4Allocator<NaIHit>* NaIHitAllocator=0;
//konstruktor
NaIHit::NaIHit(G4int modIndex, G4double enDep, G4double hitTime)
{
//wpisanie danych z konstruktora do zmiennych klasy
}
NaIHit::~NaIHit() {}
//konstuktor kopiujący
NaIHit::NaIHit(const NaIHit &right) : G4VHit()
{
   moduleIndex = right.moduleIndex;
   time = right.time;
   energyDep = right.energyDep;
}
//operator przypisania
const NaIHit& NaIHit::operator=(const NaIHit &right)
{
   moduleIndex = right.moduleIndex;
   time = right.time;
   energyDep = right.energyDep;
   return *this;
}
//operator porownania
G4int NaIHit::operator==(const NaIHit &right) const
{
  return (this==&right) ? 1 : 0;
}
//gettery i settery
```

Wykorzystanie wyników

Mamy już Sensitive Detector, który wykonuje obsługę Step-ów, interesujące dane umieszcza w obiektach typu G4VHit, powstaje kolekcja tych Hitów (uderzeń?), do której chcielibyśmy się dostać pod koniec Event-u.
Dalsza procedura jest wiec bardzo zbliżona do wersji z Prostym Licznikiem.
Modyfikujemy klasę **DetectorConstruction** oraz **EventAction** następująco:
DetectorConstruction:
Przykład wzięty z „prawdziwego" kodu:

```
if (!naISD)
{
    naISD = new NaISD("/naISD");//konstruktor
    naISD->SetModuleDeph(3); //numer kopii
}
G4SDManager* SDman = G4SDManager::GetSDMpointer();
SDman->AddNewDetector(naISD);//rejestracja w G4SDManager
photocathLogic->SetSensitiveDetector(naISD); //przypisanie do log vol
```

Wykorzystanie wyników

EventAction:

```
//1. uzyskanie informacji o ID kolekcji
G4SDManager* SDman = G4SDManager::GetSDMpointer();
int naICollID=SDman->GetCollectionID("nazwaSD/nazwaKolekcji");

//2. wyciągnij kolekcje oddziaływań dla zdarzenia (eventu)
//   (hits collections of an event)
G4HCofThisEvent* hitsCE = anEvent->GetHCofThisEvent();

//3. z hitsHC wyciągnij konkretną, interesującą kolekcję
//   (odpowiadajcą danemu ID)
NaIHitsCollection* naIHC = (NaIHitsCollection*)( hitsCE->GetHC(naICollID) );

//4. przeiteruj się po kolekcji uzyskując interesujące dane
G4int nrOfDetectors = naIHC->entries();
for(G4int i=0; i!=nrOfDetectors; i++)
{
    G4int moduleIndex = (*naIHC)[i]->GetModuleIndex();
    G4double energyDep = (*naIHC)[i]->GetEnergyDeposit();
    G4double time = (*naIHC)[i]->GetTime();
}
```

Dane można już wypisywać na ekran lub do pliku tekstowego.

Zadanie 1

Zmodyfikuj klasę NaISD tak, aby zliczała depozyt energii w danym detektorze NaI oraz czas PIERWSZEGO NIEZEROWEGO kroku w detektorze (niezerowego, czyli takiego, w którym nastąpił jakiś depozyt energii). Wypisz na ekran otrzymane wyniki, tak, aby dane z jednego eventu znajdowały się w jednej linii.