

Środowisko programistyczne GEANT4

Leszek Adamczyk

Wydział Fizyki i Informatyki Stosowanej
Akademia Górniczo-Hutnicza

Wykłady w semestrze zimowym 2013/2014

Generacja cząstek pierwotnych

- Użytkownik **musi** wyprowadzić własną klasę z klasy bazowej **G4VUserPrimaryGeneratorAction** implementując wirtualną funkcję **GeneratePrimaries()**;
- Klasa ta sama nie generuje cząstek pierwotnych steruje jedynie zewnętrznymi generatorami:
 - Konstruktor powinien dokonać instancji generatora(ów) (własnego lub wbudowanego w GEANT4);
 - Metoda **GeneratePrimaries()**
 - może randomizować niektóre wielkości używane przez generatory
 - musi wywołać funkcję **GeneratePrimaryVertex()** zaimplementowaną w generatorze.

include/MyPrimaryGeneratorAction.hh

```
class G4ParticleGun;
class G4Event;

class MyPrimaryGeneratorAction :
    public G4VUserPrimaryGeneratorAction {

public:

    // Constructor
    MyPrimaryGeneratorAction();

    // Destructor
    virtual ~MyPrimaryGeneratorAction();

    // Method
    void GeneratePrimaries(G4Event*);

private:

    // Data member
    G4ParticleGun* particleGun;

};
```

src/MyPrimaryGeneratorAction.cc

```
MyPrimaryGeneratorAction::MyPrimaryGeneratorAction()
{
    particleGun = new G4ParticleGun();
    // Inicjalizacja
    .....
}

MyPrimaryGeneratorAction::~MyPrimaryGeneratorAction()
{
    delete particleGun;
}

void MyPrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{
    // Randomizacja
    .....
    particleGun->GeneratePrimaryVertex(anEvent);
}
```

G4VPrimaryGenerator

- Informacja o cząstkach pierwotnych musi być przypisana do obiektu klasy **G4Event** przed jego procesowaniem.
- Implementując własny generator musimy zaimplementować dwie klasy których instancje są tworzone w funkcji **GeneratePrimaryVertex**
 - **G4PrimaryVertex** - określa miejsce i czas pojawienia się cząstki pierwotnej
 - **G4PrimaryParticle** - określa rodzaj cząstki, pęd początkowy, polaryzacje, ewentualnie kanały rozpadu, ...
- Każdy generator reprezentowany jest przez klasę wyprowadzoną z klasy **G4VPrimaryGenerator**.
- GEANT4 zawiera kilka gotowych implementacji:
 - prosty generator **G4ParticleGun**
 - interfejsy do zewnętrznych generatorów HEP **G4HEPEvtInterface**, **G4HEPMCInterface**
 - rozbudowany generator **G4GeneralParticleSource**

GeneralParticleSource

Generator **GeneralParticleSource**:

- Losowy wybór miejsca generacji na powierzchni zadanej bryły;
- Losowy wybór energii z zadanego widma energetycznego.
- Użyteczny do generacji cząstek ze źródeł radioaktywnych.
- Szczególnie użyteczny w zastosowaniach medycznych oraz środowiska kosmicznego
- Zaawansowane przykłady w dystrybucji GEANT'4:
[x-ray_telescope underground_physics](#)
- Szczegółowy opis
`/http://reat.space.qinetiq.com.gps/`

ParticleGun

Generator ParticleGun:

- Generuje cząstkę o zadanej energii i kierunku z danego miejsca w danym czasie;
- Wielkości te określamy za pomocą szeregu metod:
 - `SetParticleDefinition()` - rodzaj cząstki
 - `SetParticleEnergy()` - energia kinetyczna
 - `SetParticleMomentumDirection()` - kierunek
 - `SetParticlePosition()` - miejsce
 - ... polaryzacja, czas, pęd
- Wywołujemy je w konstruktorze klasy `PrimaryGeneratorAction()` jeśli nie będą ulegać zmianie.
- lub w metodzie `GeneratePrimaries` jeśli wielkości te podlegają zmianom z przypadku na przypadek;

ParticleGun: Przykład

```
MyPrimaryGeneratorAction::MyPrimaryGeneratorAction()
{
    G4int n_particle = 1;
    particleGun = new G4ParticleGun(n_particle);

    G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
    G4ParticleDefinition* particle = particleTable->FindParticle("e-");

    particleGun->SetParticleDefinition(particle);
    particleGun->SetParticleMomentumDirection(G4ThreeVector(0.,0.,1.));
    particleGun->SetParticleEnergy(1.*GeV);
    particleGun->SetParticlePosition(G4ThreeVector(0.,0.,-10.*cm));
}
```


ParticleGun: Randomizacja

W celu randomizacji generacji cząstek pierwotnych korzystamy z:

- generatora liczb pseudolosowych **G4UniformRand** zdefiniowanego w **Randomize.hh** który generuje liczby losowe z przedziału $[0,1]$ z rozkładu jednorodnego;
- klasy **HEPRandom.hh** umożliwiającej generację liczb z szeregu rozkładów np.: eksponencjalnego, normalnego, Poissona, Breita-Wignera, ...;
- wywołania odpowiedniej metody typu **Set** klasy **G4ParticleGun** w metodzie **GeneratePrimaries**

Wyniki symulacji

Mając zdefiniowaną:

- geometrię;
- listę procesów fizycznych;
- kinematykę cząstek pierwotnych

GEANT wykonuje symulacje.

Użytkownik **musi** napisać kawałek kodu aby uzyskać dostęp do informacji dla niego użytecznej. Można zrobić to na cztery sposoby:

- Korzystając z klas **G4UserTrackingAction**, **G4UserSteppingAction** i **G4UserStackingAction**.
Mamy dostęp do **pełnej** informacji ale sami **musimy** zadbać o jej przetwarzanie w trakcie procesowania przypadku.

Wyniki symulacji

- Implementując klasę **G4VSensitiveDetector**.
GEANT **automatycznie** przetwarza informację, zapisując ją w postaci obiektów klasy **G4VHit**.
Użytkownik ma dostęp do pełnego zbioru hitów (**G4THitsCollection**) po zakończeniu procesowania przypadku korzystając z klasy **G4UserEventAction**
- Korzystając z gotowej implementacji klasy **G4VSensitiveDetector** tzw. **G4MultiFunctionalDetector** opartej o obiekty **G4VPrimitiveScorer**.
"Proste liczniki" tworzą mapy, **G4THitsMap**, dostępne dla użytkownika po zakończeniu procesowania przypadku.
W GEANT mapa jest tablicą indeksowaną liczbą naturalną, będącą najczęściej numerem kopii obszaru logicznego.
HitsMap najczęściej stosuje się w aplikacjach nie wymagających informacji dla każdego przypadku a jedynie wysumowanej po wszystkich przypadkach.
- Korzystanie z wbudowanych w UI "prostych liczników" tworzących **HitsMap** na podstawie geometrii niezależnej (równoległej) do geometrii detektora.

Wyniki symulacji

- Obiekty klasy **G4VSensitiveDetector** (**G4MultiFunctionalDetector**), przypisujemy **obszarom logicznym**.
- Tworząc geometrię detektora, czasami należy drobniej "podzielić" obszar detektora niż by to wynikało z jego budowy fizycznej. Np. aby uzyskać informacje z mniejszych obszarów.
- Powinniśmy korzystać z wielokrotnego umieszczania tego samego obszaru logicznego (**Parameterised, Replica**), wtedy używamy jednego obiektu klasy **SensitiveDetector** a jego **HitsMap** w naturalny sposób indeksowana jest numerem kopii tego obszaru.

MultiFunctionalDetector

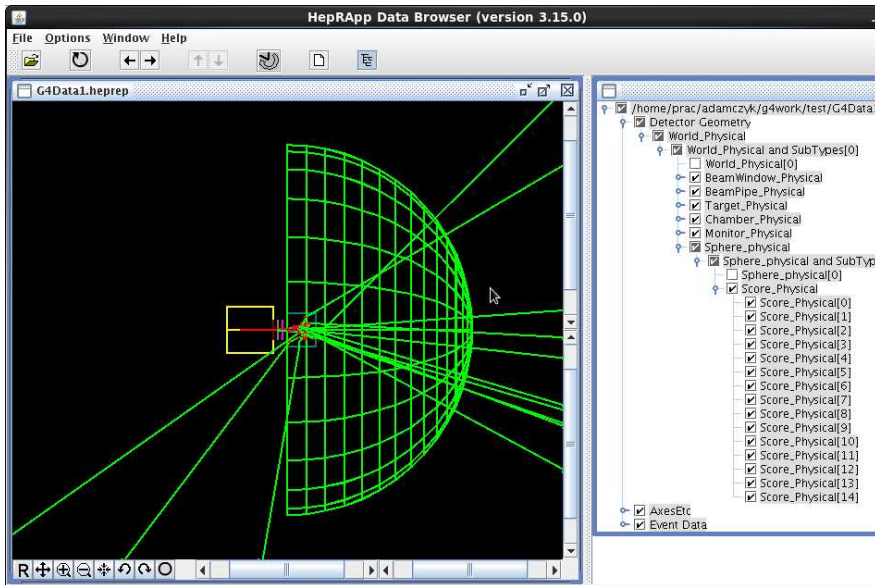
- **G4MultiFunctionalDetector** jest implementacją klasy bazowej **G4VSensitiveDetector**.
- Zamiast samemu implementować **SensitiveDetector** budujemy go rejestrując obiekty klasy **PrimitiveScorer** do **MultiFunctionalDetector**.
- **G4VPrimitiveScorer** jest klasą bazową z której wyprowadzamy konkretne klasy pochodne. Istnieje szereg gotowych klas np: **G4PSEnergyDeposit** lub **G4PSTrackLength**.
- Pełna lista w rozdziale 4.4.6 (Application Developers Guide)

MultiFunctionalDetector

Aby korzystać z funkcjonalności **MultiFunctionalDetector** należy:

- Utworzyć obiekt klasy **G4MultiFunctionalDetector**;
- Zarejestrować go do **SensitiveDetectorManager**'a (obiekt klasy **G4SDManager**);
- Przypisać **MultiFunctionalDetector** do jednego lub kilku obszarów logicznych;
- Utworzyć obiekt lub kilka obiektów klasy **PrimitiveScorer**;
- Zarejestrować **PrimitiveScorer's** do **MultiFunctionalDetector**

Aplikacja My



Aplikacja My

```
// Create a new sensitive detector named "MyDetector"  
G4MultiFunctionalDetector* detector =  
    new G4MultiFunctionalDetector("MyDetector");
```

```
// Get pointer to detector manager  
G4SDManager* manager = G4SDManager::GetSDMpointer();
```

```
// Register detector with manager  
manager->AddNewDetector(detector);
```

```
// Attach detector to scoring volume  
scoreLogical->SetSensitiveDetector(detector);
```

```
// Create a primitive Scorer named MyScorer  
G4PSSphereSurfaceCurrent* scorer =  
    new G4PSSphereSurfaceCurrent("MyScorer", fCurrent_In);
```

```
// Register scorer with detector  
detector->RegisterPrimitive(scorer);
```


G4THitsMap

- Każdy **G4VPrimitiveScorer** generuje jedną mapę **G4THitsMap<G4double>** na przypadek.
- Mapa ta jest tablicą o wartościach **G4double** **indeksowaną numerem kopii** obszaru logicznego do którego przypisany jest **MultiFunctionalDetector**.
- Mapa może być indeksowana numerem kopii matki (babci, itd).
- Trzeci parametr konstruktora obiektu **PrimitiveScorer** określa którego przodka numer kopii będzie indeksem mapy.

```
G4PSSphereSurfaceCurrent(..., ..., G4int depth=0);
```

- W bardziej skomplikowanych schematach indeksowania można zaimplementować własną metodę **GetIndex()**

G4VSDFilter

- Obiekt klasy **SDFilter** może być przypisany do **PrimitiveScorer** w celu określenia na jaki rodzaj cząstek dany licznik jest czuły.
Istnieją cztery gotowe implementacje klasy **G4VSDFilter**:
 - **G4SDChargedFilter**
 - **G4SDNeutralFilter**
 - **G4SDParticleFilter**
 - **G4SDKineticEnergyFilter**

```
G4VSDFilter* gammaFilter =  
    new G4SDParticleFilter("gammaFilter");  
gammaFilter->Add("gamma");  
scorer->SetFilter(gammaFilter);
```

G4HCofThisEvent

- **G4HCofThisEvent** jest klasą która zawiera **HitsCollection** oraz **HitsMap** dla danego przypadku.
- **HitsMap** jest dostępna z **HCofThisEvent** poprzez unikalny **numer identyfikacyjny** danej **HitsMap**.
- Numer identyfikacyjny można uzyskać poprzez funkcję **G4SDManager::GetCollectionID()**

```
G4SDManager* manager = G4SDManager::GetSDMpointer();
G4int mapID = manager->
    GetCollectionID("MyDetector/MyScorer")

//    G4Event* evt
G4HCofThisEvent* hce = evt->GetHCofThisEvent();
G4THitsMap<double>* eventGammaScorer =
    (G4THitsMap<double>*) (hce->GetHC(mapID));
```

G4THitsMap

G4THitsMap<G4double> ma wbudowane operatory:

- operator indeksowania `map[index]`
zwraca wskaźnik do elementu mapy o indeksie `index`
Wskaźnik o wartości zero oznacza że nie istnieje element mapy o indeksie `index`
- `map1 + = map2`
Dodaje wartości mapy `map2` do odpowiednich wartości mapy `map1`.
Usprawnia akumulacje danych po wszystkich przypadkach.

Przykład

```
G4SDManager* manager = G4SDManager::GetSDMpointer();
G4int mapID = manager->
    GetCollectionID("MyDetector/MyScorer")

//   G4Event* evt
G4HCofThisEvent* hce = evt->GetHCofThisEvent();
G4THitsMap<double>* eventGammaScorer =
    (G4THitsMap<double>*) (hce->GetHC(mapID));
```

```
//   G4THitsMap<G4double> runGammaScorer

runGammaScorer += *eventGammaScorer;
```

```
for (G4int i=0; i<runGammaScorer.GetSize(); i++) {
    G4cout<< runGammaScorer[i] << G4endl;
}
```

G4UserRunAction

- Po zakończeniu procesowania przypadku obiekt klasy **G4Event** posiada informację o **HitsMap** i **HitsCollection** dla danego przypadku.
- Użytkownik **musi** napisać kawałek kodu aby np. sumować wartości **HitsMap** lub zapisywać je do dalszej analizy.
- Można to zrobić na wiele sposobów;
- Na warsztatach zaimplementujemy własne klasy **G4UserRunAction** oraz **G4Run** .
- Klasa **G4UserRunAction** ma następujące metody wirtualne:
 - **BeginOfRunAction()**
 - **EndOfRunAction()**
 - **GenerateRun()** - służy do utworzenia obiektu samodzielnie zaimplementowanej klasy **G4Run**

G4Run

- Klasa reprezentująca zbiór przypadków.
- Obiekt tej klasy jest tworzony i niszczone przez **RunManagera** (bezpośrednio lub za pośrednictwem funkcji **GenerateRun()** klasy **G4UserRunAction**).
- Funkcja składowa **G4Run::RecordEvent** jest automatycznie wywoływana przez **RunManagera** po zakończeniu procesowania każdego przypadku.
- W tej funkcji mamy dostęp do **HitsMap** i **HitsCollection** po każdym przypadku.

G4VHitsCollection

Klasa **G4VHitsCollection** jest klasą bazową z której wyprowadzono klasy szablone **G4THitsCollection** oraz **G4THitsMap**

- **G4THitsCollection** jest szablonoową klasą pojemnikową typu **vector** służącą do przechowywania wskaźników do obiektów jednej konkretnej klasy wyprowadzonej z klasy **G4VHit** (hitów)
- **G4THitsMap** jest szablonoową klasą pojemnikową typu **map** służącą do przechowywania wskaźników do obiektów jednej konkretnej (ale dowolnej) klasy indeksowanej dowolną wartością. Mogą to być hity ale nie muszą np. **PrimitiveScorer** używają **G4THitsMap** które zawierają wskaźniki do wartości **G4double** indeksowane liczbą naturalną.

G4VHit

- Klasa bazowa **G4VHit** służy do wyprowadzania własnych klas **hitów** zawierających potrzebną nam informację np.:
 - pozycje i czas kroku (G4Step)
 - pęd i energie cząstek (G4Track)
 - depozyty energii podczas kroku (G4Step)
 - informacje geometryczne
- **Hit** przechowywany jest w dedykowanej kolekcji która jest instancją klasy szablonej **G4THitsCollection**
- **HitsCollection** dostępna jest z obiektu klasy **G4Event** poprzez obiekt klasy **G4HCOfThisEvent**

Diagram klas

Diagram klas



konkretna klasa GEANT4



wirtualna klasa bazowa GEANT4

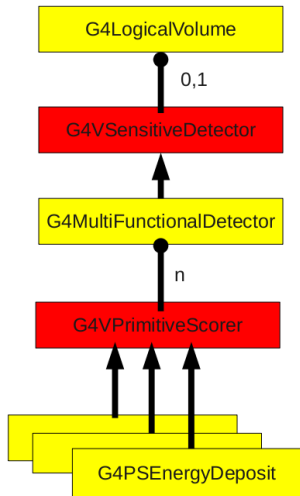


Diagram klas

Diagram klas

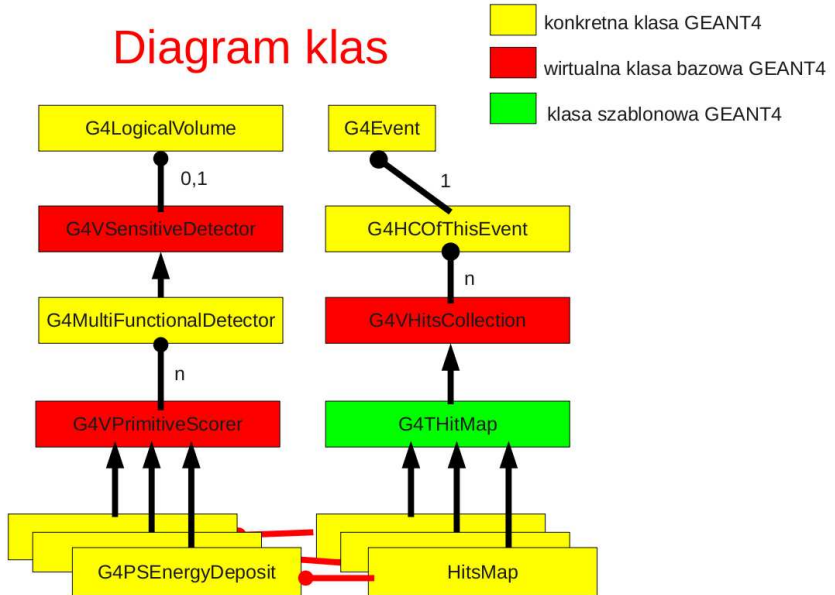


Diagram klas

Diagram klas

