# Zastosowanie pakietu Geant4 w fizyce jądrowej Wykład 6

Aleksandra Fijałkowska

2 grudnia 2021

# Procesy fizyczne

W naszym projekcie oprócz poruszonych do tej pory klas znajduje się także klasa PhysicsList. Implementacja tej klasy jest dość zagmatwana, zacznę więc od informacji ogólnych a potem przejdziemy do szczegółów w kodzie. Kluczowe są trzy metody:

- **virtual void ConstructParticle()** – służy określeniu (zbudowaniu) możliwych cząstek
- **virtual void ConstructProcess()** – określa procesy fizyczne (dla każdej z cząstek)
- **virtual void SetCuts()** – określa warunki kreacja cząstek wtórnych (metoda nie jest czysto wirtualna, a wiec nie musi być implementowana)

# **ConstructParticle()**, czyli definiowanie cząstek

Każda cząstka reprezentowana jest przez osobną klasę wywiedzioną z klasy bazowej **G4ParticleDefinition**. Wśród dostępnych cząstek znajdują się:

- leptony ($e$, $\mu$, $\tau$ oraz odpowiadające im neutrina)
- hadrony (mezony, jak np. $\pi$, $K$, $J$ itd.; bariony – proton, neutron, $\Lambda$, $\Omega$; jony)
- bozony (fotony, fotony optyczne, geantino)
- stany wzbudzone (mezonów, jonów)
- kwarki, gluony (nigdy nie korzystałam)

Zbudowanie cząstki polega na wywołaniu jednaj z trzech statycznych metod, powodujących zwrócenie instancji klasy reprezentującej tę cząstkę. Klasy te są singletonami, czyli posiadają tylko jedną instancję w całym programie.

# Przykład **G4Electron**

```
class G4Electron : public G4ParticleDefinition
{
 private:
   static G4Electron* theInstance;
   G4Electron(){}
   ~G4Electron(){}

 public:
   static G4Electron* Definition();
   static G4Electron* ElectronDefinition();
   static G4Electron* Electron();
};
```

Wszystkie trzy statyczne, ubliczne metody robią DOKŁADNIE to samo, mogą być więc wywoływane zamiennie (uwaga na przykłady).

```
void PhysicsList::ConstructParticle()
{
  // leptons
  G4Electron::ElectronDefinition();
  G4Positron::PositronDefinition();
  G4NeutrinoE::NeutrinoEDefinition();
  G4AntiNeutrinoE::AntiNeutrinoEDefinition();
}
```

Analogicznie tworzy się inne, potrzebne cząstki.

# Aktywacja cząstek, szybsza ścieżka

Jeśli nie chcemy każdej z cząstek tworzyć osobno możemy posłużyć się pomocniczymi klasami (konstruktorami), które budują wszystkie cząstki wybranego typu (np leptony).
Przykład z naszego projektu:

```
void EMPhysics::ConstructParticle()
{
  G4BosonConstructor bConstructor;
  bConstructor.ConstructParticle();

  G4LeptonConstructor lConstructor;
  lConstructor.ConstructParticle();
...
  G4IonConstructor iConstructor;
  iConstructor.ConstructParticle();
}
```

Jak wygląda przykładowa metoda **ConstructParticle()**?

```
void G4BosonConstructor::ConstructParticle()
{
  // pseudo-particles
  G4Geantino::GeantinoDefinition();
  G4ChargedGeantino::ChargedGeantinoDefinition();

  // gamma
  G4Gamma::GammaDefinition();

  // optical photon
  G4OpticalPhoton::OpticalPhotonDefinition();
}
```

# Lista dostępnych cząstek

Zbiór zbudowanych cząstek (aktywnych) zawiera klasa **G4ParticleTable**.
Tablica jest singletonem, dlatego dostęp do niej odbywa się przez wywołanie statycznej, publicznej metody **GetParticleTable()**, która zwraca wskaźnik do obiektu: **G4ParticleTable* G4ParticleTable::GetParticleTable()**.
Klasa ma szereg getterów zwracających instancję klasy reprezentującej cząstkę.

```
G4ParticleDefinition *  FindParticle (const G4String &particle_name)
G4ParticleDefinition *  FindAntiParticle (const G4String &particle_name)
G4ParticleDefinition *  GetIon (G4int atomicNumber,
                                G4int atomicMass,
                                G4double excitationEnergy)
```

Z **G4ParticleTable** korzystaliśmy w klasie **PrimaryGeneratorAction**.

# ConstructProcess(), czyli określenie procesów fizycznych



Procesem nazywamy sposób oddziaływania cząstek z materią. Procesy mogą być dyskretne (jak np. rozproszenie Comptona) lub ciągłe (jonizacja). Wszystkie procesy dziedziczą po klasie bazowej **G4VProcess**, ta zaś posiada szereg czysto wirtualnych metod, które muszą być zaimplementowane w klasach pochodnych:

```
virtual G4VParticleChange*  PostStepDoIt
virtual G4VParticleChange*  AlongStepDoIt
virtual G4VParticleChange*  AtRestDoIt
virtual G4double AlongStepGetPhysicalInteractionLength
virtual G4double AtRestGetPhysicalInteractionLength
virtual G4double PostStepGetPhysicalInteractionLength
```

Deweloper może napisać swój własny proces (implementując powyższe metody) i przyporządkować mu pewne cząstki. Własne procesy są traktowane na równi z "wbudowanymi" podczas przeprowadzania cząstki przez materię.

My na zajęciach będziemy wykorzystywać tylko procesy wbudowane w bibliotekę.

# ConstructProcess(), czyli określenie procesów fizycznych

Biblioteka Geant4 oferuje szereg wbudowanych procesów fizycznych różnych typów:

- elektromagnetyczne
- hadronowe
- rozpadu
- optyczne
- transportu
- parametryzowane

# Procesy elektromagnetyczne

- Fotony
  - Efekt fotoelektryczny (**G4PhotoElectricEffect**)
  - Rozproszenie Comptona (**G4ComptonScattering**)
  - Kreacja pary elektron-pozyton (**G4GammaConversion**)
  - Rozproszenie Rayleigh-a (**G4RayleighScattering**))
  - Kreacja pary $\mu^+$-$\mu^-$ (**G4GammaConversionToMuons**)
- Elektrony/pozytony
  - Jonizacja (**G4eIonisation**)
  - Bremsstrahlung – promieniowanie hamowania (**G4eBremsstrahlung**)
  - Rozproszenie (**G4eMultipleScattering**)
  - Anihilacja pozytonu + emisja dwóch kwantów promieniowania $\gamma$ (**G4eplusAnnihilation**)
  - Anihilacja pozytonu do dwóch mionów (**G4AnnihiToMuPair**)
  - Anihilacja pozytonu do dwóch hadronów (**G4eeToHadrons**)
- Miony
  - Jonizacja (**G4MuIonisation**)
  - Promieniowanie hamowania (**G4MuBremsstrahlung**)
  - Kreacja pary $e^+$ $e^-$ (**G4MuPairProduction**)
  - Rozproszenie (**G4MuMultipleScattering**)

# Procesy elektromagnetyczne

- Hadrony i jony
  - Jonizacja (**G4hIonisation**)
  - Jonizacja dla jonów (**G4ionIonisation**)
  - Rozproszenie (**G4hMultipleScattering**)
  - Promieniowanie hamowania (**G4hBremsstrahlung**)
  - Kreacja pary $e^+$ $e^-$ (**G4hPairProduction**)
- Promieniowanie X oraz fotony optyczne (powstanie)
  - Promieniowanie synchrotronowe (**G4hIonisation**)
  - Promieniowanie przejścia (**G4TransitionRadiation**)
  - Promieniowanie Czerenkowa (**G4Cerenkov**)
  - Scyntylacja (**G4Scintillation**)

# Procesy hadronowe i rozpady

Procesy hadronowe:

- Rozproszenie elastyczne
- Rozproszenie nieelastyczne
- Wychwyt
- Rozszczepienie
- Rozpady silne (np. jądrowe)
- Reakcje fotojądrowe

Rozpady:

- Rozpady słabe (np. przemiana $\beta$)
- Rozpady elektromagnetyczne
- Rozpady silne (już wspomniane)

# Procesy optyczne

Procesy optyczne:

- Absorpcja
- Rozproszenie Rayleigh-a
- Procesy przejścia przez granicę dwóch ośrodków (dielektryk-dielektryk, dielektryk-metal, dielektryk-cialo czarne)
- Wavelength Shifting (WLS) – przesunięcie długości fali

Wszystkie procesy są szczegółowo opisane w Physics Reference Manual, dostępny na stronie http://geant4-userdoc.web.cern.ch/geant4-userdoc/UsersGuides/PhysicsReferenceManual/fo/PhysicsReferenceManual.pdf

# Implementacja

Klasa PhysicsList może być wywiedziona z klasy bazowej **G4VUserPhysicsList**. Obowiązkiem dewelopera jest zaimplenetowanie czysto wirtualnych metod **ConstructParticle()** (omówionę uprzednio) oraz **ConstructProcess()**. **ConstructProcess()** jest miejscem, w którym rejestrujemy procesy fizyczne odpowiadające uprzednio utworzonym cząstkom.

```
void PhysicsList::ConstructProcess()
{
  AddTransportation();//transposr musi być dodany zawsze
  //przykład procesów EM
  auto aParticleIterator=GetParticleIterator();
  aParticleIterator->reset();
  while( (*aParticleIterator)() )
  {
    G4ParticleDefinition* particle = aParticleIterator->value();
    G4ProcessManager* pmanager = particle->GetProcessManager();
    G4String particleName = particle->GetParticleName();

    if (particleName == "gamma") {
      pmanager->AddDiscreteProcess(new G4GammaConversion());
      pmanager->AddDiscreteProcess(new G4ComptonScattering());
      pmanager->AddDiscreteProcess(new G4PhotoElectricEffect());
      pmanager->AddDiscreteProcess(new G4RayleighScattering());
    } else if (particleName == "e+") {
      pmanager->AddProcess(new G4eMultipleScattering(),-1, 1, 1);
      pmanager->AddProcess(new G4eIonisation(),       -1, 2, 2);
      pmanager->AddProcess(new G4eBremsstrahlung(),   -1, 3, 3);
      pmanager->AddProcess(new G4eplusAnnihilation(),  0,-1, 4);
    }
    ...
  }
}
```

# Implementacja

W celu zachowania przejrzystości zachęcam do stworzenia osobnych metod wprowadzających poszczególne typy oddziaływań.
Innym rozwiązaniem (zastosowanym w naszym przykładzie) jest dziedziczenie po **G4VModularPhysicsList**.
W takiej sytuacji w konstruktorze klasy **PhysicsList** „rejestrujemy" obiekty reprezentujące kolejne rodzaje oddziaływań (metoda **RegisterPhysics**).
Argumentem PhysicsList jest wskaźnik do obiektów wywiedzionych z klasy bazowej **G4VPhysicsConstructor**, która podobnie do **G4VUserPhysicsList** posiada wirtualne metody **ConstructParticle()** i **ConstructProcess()**, które należy zaimplemenować w sposób analogiczny.

Wybranie drugiej ścieżki zwalnia nas z obowiązku dodawania procesu transportu (jest on dodany automatycznie).

# Listy fizyczne

Biblioteka Geant4 oferuje gotowe listy fizyczne, obejmujące zarówno produkcję cząstek, jak i rejestrację procesów fizycznych. Niektórzy rekomendują wykorzystanie list fizycznych na początku przygody z pakietem Geant4.
Dostępne listy fizyczne można znaleźć pod adresem:
https://geant4.web.cern.ch/node/155 oraz
http://irfu.cea.fr/dphn/Spallation/physlist.html
Przykład dołączenia procesów fizycznych w formie gotowej listy:

```
G4int verbose = 1;
FTFP_BERT* physlist = new FTFP_BERT(verbose);
runManager->SetUserInitialization(physlist);
```

```
G4int verbose = 1;
G4PhysListFactory factory;
G4VModularPhysicsList* physlist = factory.GetReferencePhysList("FTFP_BERT_EMV");
physlist.SetVerboseLevel(verbose);
runManager->SetUserInitialization(physlist);
```

Proszę zwrócić uwagę, że poniższe fragmenty kodu powinny zostać umieszczone w funkcji „main()”.

# Próg powstania cząstek wtórnych – **SetCut()**

Każdy proces ma wewnętrzny próg powstawania cząstek wtórnych, każda cząstka jest „trakowana” do końca zasięgu, określonego właśnie w metodzie **SetCut()**.
Użytkownik określa minimalny ZASIĘG cząstki, a nie jej energię. Ta jest wyznaczana w oparciu o zadany zasięg indywidualnie dla każdego materiału.
Domyślną wartością progu dla wszystkich cząstek jest 1,0 mm.

# Uwaga ogólna

Każdy projekt wymaga osobnego przemyślenia potrzebnych procesów fizycznych.
Oprócz procesów Geant4 oferuje różne modele mniej lub bardziej adekwatnych do naszego problemu (np. NeutronHP).

Dobrym początkiem przygody jest skorzystanie z gotowej listy lub poszukanie przykładu zbliżonego do naszych potrzeb.