Kalorymetr wysokiej rozdzielczości (HRC- High Resolution Calorymeter) Środowisko programistyczne Geant4 Semestr Zimowy 2024/2025

Piotrowski Dawid
Informatyka Stosowana
Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

13.02.2025

Spis treści

Streszczenie

Niniejsze sprawozdanie przedstawia symulację działania kalorymetru wysokiej rozdzielczości typu spagetti, zrealizowaną w środowisku Geant4. Głównym celem symulacji jest zmierzenie energii oraz położenia padającego na detektor wysokoenergetycznego fotonu. Foton, przechodząc przez absorber wykonany z wolframu, generuje kaskadę elektromagnetyczną, w wyniku której powstają fotony optyczne w wiązkach kwarcu. Dane (numer zdarzenia, energia oraz czas dotarcia fotonów) są zapisywane do ntupla, a ich analiza (przy użyciu makra ROOT) pozwala na odtworzenie rozkładów istotnych parametrów.

Rozdział 1

Wstęp

Kalorymetr wysokiej rozdzielczości (HRC) stanowi istotny element detekcji cząstek w eksperymentach wysokiej energii. Projekt, którego dotyczy sprawozdanie, polega na symulacji detekcji fotonu o energii 100 MeV, który po przejściu przez absorber generuje kaskadę elektromagnetyczną. W wyniku oddziaływań w absorberze powstają wtórne cząstki, w tym fotony optyczne, rejestrowane przez włókna kwarcowe. Uzyskane dane, po odpowiedniej analizie, umożliwiają kalibrację detektora oraz wyznaczenie położenia padającego fotonu.

Rozdział 2

Opis projektu

Projekt realizowany jest w środowisku Geant4 i dotyczy symulacji kalorymetru wysokiej rozdzielczości typu spagetti. Główne elementy projektu to:

- **Absorber**: Blok wykonany z wolframu o wymiarach $1 \times 1 \times 50\,\mathrm{cm}^3$, w którym generowana jest kaskada elektromagnetyczna.
- **Włókna kwarcowe**: Cylindry o promieniu 0.625 mm i długości 50 cm, zanurzone w absorberze. Każde włókno posiada otoczkę z PMMA o grubości 0.1 mm.
- **Układ włókien**: Implementacja 9 włókien rozmieszczonych w regularnej siatce, gdzie oś centralnego włókna pokrywa się z osią absorbera.
- **Pre-absorber**: Dodatkowy blok miedziany o grubości 10 cm umieszczony przed głównym absorberen.

Realizowana symulacja opiera się na generacji pierwotnego fotonu o energii 100 MeV, który po interakcji w absorberze generuje kaskadę cząstek. Foton optyczny, docierający do końca włókna, jest rejestrowany i zapisywany do ntupla w postaci: numer zdarzenia, energia oraz czas dotarcia.

Rozdział 3

Implementacja symulacji

Symulacja w Geant4 została podzielona na kilka głównych komponentów:

3.1 Definicja detektora

Kod odpowiedzialny za konstrukcję geometrii detektora znajduje się w plikach `MyDetectorConstruction.hh` oraz `MyDetectorConstruction.cc`. W ramach konstrukcji definiowane są:

- Świat symulacji (World),
- Absorber (blok z wolframu),
- Pre-absorber (blok miedziany),
- Włókna kwarcowe oraz ich otoczka z PMMA.

3.2 Lista procesów fizycznych

Implementacja listy fizycznej, zawierającej procesy elektromagnetyczne, hadronowe oraz optyczne (w tym promieniowanie Czerenkowa), znajduje się w plikach `MyPhysicsList.hh` i `MyPhysicsList.cc`.

3.3 Generator cząstek pierwotnych

Pliki `MyPrimaryGeneratorAction.hh` oraz `MyPrimaryGeneratorAction.cc` odpowiadają za generowanie pierwotnej cząstki (foton o energii 100 MeV) skierowanej na detektor.

3.4 Akcje symulacyjne

W plikach `MyRunAction.hh`/`MyRunAction.cc` oraz `MySteppingAction.hh`/`MySteppingAction.cc` zrealizowano:

- Inicjalizację symulacji, tworzenie ntupla oraz zapisywanie danych (RunAction),
- Śledzenie kroków symulacji i rejestrację fotonów optycznych docierających do końca włókna (SteppingAction).

Rozdział 4

Analiza danych

Dane z symulacji są zapisywane do pliku ROOT (`MyOutput.root`) w postaci ntupla, który zawiera:

- Numer zdarzenia,
- Energię fotonu (w MeV),
- Czas dotarcia fotonu (w ns).

Do analizy danych wykorzystano makro ROOT, które automatycznie dopasowuje zakresy histogramów do danych. Poniżej zamieszczono przykładową implementację makra:

```
void analyzePhotons() {
  // Otworz plik z danymi
  TFile* file = TFile::Open("MyOutput.root");
  if (!file || file->IsZombie()) {
    std::cout << "Error: Could not open file MyOutput.root" << std::endl;
    return;
  }
  // Pobierz drzewo ntupla "Photons"
  TTree* tree = (TTree*)file->Get("Photons");
  if (!tree) {
    std::cout << "Error: TTree 'Photons' not found in the file." << std::endl;
    return;
  }
  // Zmienne do odczytu danych
  Double_t energy = 0, time = 0;
  tree->SetBranchAddress("Energy", &energy);
  tree->SetBranchAddress("Time", &time);

  // Ustalanie zakresu dla energii
  Double_t minEnergy = tree->GetMinimum("Energy");
  Double_t maxEnergy = tree->GetMaximum("Energy");
  if (minEnergy == maxEnergy) { minEnergy -= 1; maxEnergy += 1; }
  else { Double_t marginE = 0.1*(maxEnergy-minEnergy); minEnergy -= marginE; maxEnergy +=
     marginE; }

  // Ustalanie zakresu dla czasu
  Double_t minTime = tree->GetMinimum("Time");
  Double_t maxTime = tree->GetMaximum("Time");
  if (minTime == maxTime) { minTime -= 1; maxTime += 1; }
  else { Double_t marginT = 0.1*(maxTime-minTime); minTime -= marginT; maxTime += marginT
    ; }

  // Histogram rozkladu energii
  TH1D* hEnergy = new TH1D("hEnergy", "Optical Photons Energy Distribution", 100,
    minEnergy, maxEnergy);
  hEnergy->GetXaxis()->SetTitle("Energy (MeV)");
  hEnergy->GetYaxis()->SetTitle("Number of Photons");

  // Histogram rozkladu czasu
  TH1D* hTime = new TH1D("hTime", "Optical Photons Time Distribution", 100, minTime,
    maxTime);
  hTime->GetXaxis()->SetTitle("Time (ns)");
  hTime->GetYaxis()->SetTitle("Number of Photons");

  // Wypelnianie histogramow
  Long64_t nEntries = tree->GetEntries();
  for (Long64_t i = 0; i < nEntries; ++i) {
    tree->GetEntry(i);
    hEnergy->Fill(energy);
```

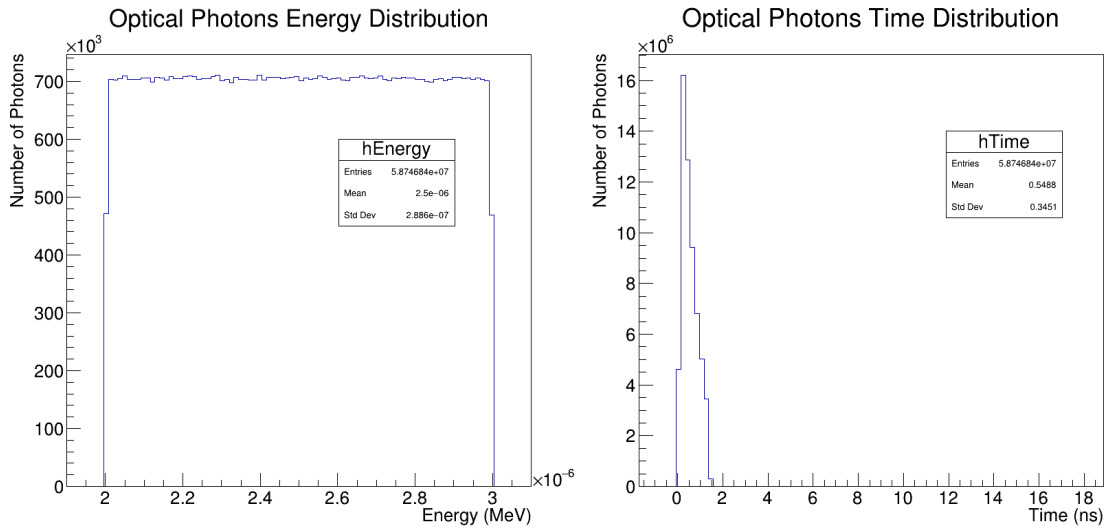
```
    hTime->Fill(time);
  }

  // Rysowanie histogramow na canvasie
  TCanvas* c = new TCanvas("c", "Optical Photons Histograms", 1200, 600);
  c->Divide(2, 1);
  c->cd(1); hEnergy->Draw();
  c->cd(2); hTime->Draw();
  c->SaveAs("PhotonHistograms.png");
}
```

Listing 4.1: Makro analizy ROOT (analyzePhotons.C)

4.1 Wyniki analizy

Na rysunku 4.1 zaprezentowano histogramy rozkładów energii (po lewej) oraz czasu (po prawej) fotonów optycznych, które dotarły do końca włókna.



Rysunek 4.1: Histogramy energii i czasu fotonów optycznych.

W przypadku **rozkładu energii** widać, że zakres energii fotonów zawiera się w przedziale około 2–3 eV (co odpowiada wartościom rzędu 2×10^{-6}–3×10^{-6} MeV). Średnia energia wynosi w przybliżeniu 2.5×10^{-6} MeV (ok. 2.5 eV), a odchylenie standardowe to około 2.9×10^{-7} MeV. Równomierne rozłożenie liczby fotonów w tym obszarze jest charakterystyczne dla promieniowania Czerenkowa w kwarcu.

Z kolei **rozkład czasu** pokazuje, że większość fotonów dociera do detektora w przedziale 0–2 ns, ze średnim czasem 0.55 ns i odchyleniem standardowym 0.35 ns. Widoczna jest również niewielka liczba fotonów o dłuższych czasach, sięgających nawet kilkunastu nanosekund. Zjawisko to można tłumaczyć odbiciami fotonów wewnątrz włókna czy różnymi drogami przebytymi przez cząstki przed dotarciem do końca.

Łącznie w zarejestrowanych danych widzimy ponad 5.8×10^7 fotonów optycznych. Tak duża liczba detekcji potwierdza intensywny charakter promieniowania Czerenkowa w symulowanym układzie. Szybki czas dotarcia fotonów (rzędu setnych części nanosekundy) sugeruje również, że w modelu poprawnie odwzorowano własności optyczne kwarcu i geometrię detektora.

Rozdział 5

Wnioski

Przeprowadzona symulacja umożliwiła:

- Pomiar energii pierwotnego fotonu na podstawie liczby fotonów optycznych,
- Wyznaczenie położenia punktu wejścia fotonu poprzez analizę poprzecznego profilu kaskady.

Chociaż symulacja obejmuje jedynie centralne włókno, uzyskane wyniki stanowią solidną podstawę do dalszej rozbudowy modelu, np. o implementację pełnego układu kilkudziesięciu włókien.

5.1 Dostępność kodu źródłowego

Wszelkie materiały użyte w projekcie, łącznie z kodem symulacji i skryptami analitycznymi, dostępne są w repozytorium GitHub pod adresem:

`https://github.com/LeoTheOriginal/Geant4-programming-environment/tree/main`

Dodatek A

Kod źródłowy

Poniżej zamieszczono fragmenty kluczowych plików źródłowych projektu. W celu pełnej analizy kodu można dodać kolejne pliki.

A.1 MyDetectorConstruction

```
#include "MyDetectorConstruction.hh"
#include "G4Box.hh"
#include "G4Colour.hh"
#include "G4LogicalVolume.hh"
#include "G4Material.hh"
#include "G4NistManager.hh"
#include "G4PVPlacement.hh"
#include "G4SystemOfUnits.hh"
#include "G4Tubs.hh"
#include "G4VisAttributes.hh"
#include "G4OpticalSurface.hh"

MyDetectorConstruction::MyDetectorConstruction()
    : fpWorldLogical(nullptr), fpWorldPhysical(nullptr) {}

MyDetectorConstruction::~MyDetectorConstruction() {}

G4VPhysicalVolume* MyDetectorConstruction::Construct() {
  DefineMaterials();
  SetupGeometry();
  return fpWorldPhysical;
}

void MyDetectorConstruction::DefineMaterials() {
  G4NistManager* nistManager = G4NistManager::Instance();

  // Define materials
  G4Material* air = nistManager->FindOrBuildMaterial("G4_AIR");
  G4Material* copper = nistManager->FindOrBuildMaterial("G4_Cu");
  G4Material* quartz = nistManager->FindOrBuildMaterial("G4_SILICON_DIOXIDE");
  G4Material* pmma = nistManager->FindOrBuildMaterial("G4_PLEXIGLASS");
  G4Material* tungsten = nistManager->FindOrBuildMaterial("G4_W");

  // Optical properties for Quartz (SiO2)
  G4MaterialPropertiesTable* mptQuartz = new G4MaterialPropertiesTable();
  G4double photonEnergy[] = {2.0*eV, 3.0*eV}; // Example energy range
  G4double rindexQuartz[] = {1.54, 1.54}; // Refractive index
  G4double absorptionQuartz[] = {50.*cm, 50.*cm}; // Absorption length
  mptQuartz->AddProperty("RINDEX", photonEnergy, rindexQuartz, 2);
  mptQuartz->AddProperty("ABSLENGTH", photonEnergy, absorptionQuartz, 2);
  quartz->SetMaterialPropertiesTable(mptQuartz);

  // Optical properties for PMMA (Cladding)
  G4MaterialPropertiesTable* mptPMMA = new G4MaterialPropertiesTable();
  G4double rindexPMMA[] = {1.49, 1.49};
  G4double absorptionPMMA[] = {50.*cm, 50.*cm};
  mptPMMA->AddProperty("RINDEX", photonEnergy, rindexPMMA, 2);
  mptPMMA->AddProperty("ABSLENGTH", photonEnergy, absorptionPMMA, 2);
  pmma->SetMaterialPropertiesTable(mptPMMA);

  // --- Optical properties for Tungsten ---
  G4double rindexTungsten[] = {3.3, 3.3};  // Refractive index
  G4double absorptionTungsten[] = {10.*cm, 10.*cm};  // Absorption length
```

```
54
55    G4MaterialPropertiesTable* mptTungsten = new G4MaterialPropertiesTable();
56    mptTungsten->AddProperty("RINDEX", photonEnergy, rindexTungsten, 2);
57    mptTungsten->AddProperty("ABSLENGTH", photonEnergy, absorptionTungsten, 2);
58    tungsten->SetMaterialPropertiesTable(mptTungsten);
59
60    // --- Surface properties ---
61    G4OpticalSurface* opticalSurface = new G4OpticalSurface("FiberSurface");
62    opticalSurface->SetType(dielectric_metal);
63    opticalSurface->SetModel(unified);
64    opticalSurface->SetFinish(polished);
65
66    G4MaterialPropertiesTable* surfaceMPT = new G4MaterialPropertiesTable();
67    G4double reflectivity[] = {0.9, 0.9};  // Odbicie
68    surfaceMPT->AddProperty("REFLECTIVITY", photonEnergy, reflectivity, 2);
69    opticalSurface->SetMaterialPropertiesTable(surfaceMPT);
70
71    // Print material information
72    G4cout << *(G4Material::GetMaterialTable()) << G4endl;
73  }
74
75  void MyDetectorConstruction::SetupGeometry() {
76    // World
77    G4double worldSizeXY = 1.0 * m;
78    G4double worldSizeZ = 3.0 * m;
79    G4Box* worldSolid =
80        new G4Box("World", worldSizeXY / 2, worldSizeXY / 2, worldSizeZ / 2);
81    fpWorldLogical = new G4LogicalVolume(worldSolid, G4Material::GetMaterial("G4_AIR"),
82                                         "World");
83    fpWorldPhysical = new G4PVPlacement(
84        nullptr, G4ThreeVector(), fpWorldLogical, "World", nullptr, false, 0);
85
86    // Absorber (Tungsten)
87    G4double absorberSizeXY = 1.0 * cm;
88    G4double absorberSizeZ = 50.0 * cm;
89    G4Box* absorberSolid = new G4Box("Absorber", absorberSizeXY / 2,
90                                     absorberSizeXY / 2, absorberSizeZ / 2);
91    G4LogicalVolume* absorberLogical =
92        new G4LogicalVolume(absorberSolid, G4Material::GetMaterial("G4_W"),
93                            "Absorber");
94    new G4PVPlacement(nullptr, G4ThreeVector(), absorberLogical, "Absorber",
95                      fpWorldLogical, false, 0);
96
97    // Pre-absorber (Copper)
98    G4double preAbsorberThickness = 10.0 * cm;
99    G4Box* preAbsorberSolid =
100       new G4Box("PreAbsorber", absorberSizeXY / 2, absorberSizeXY / 2,
101                 preAbsorberThickness / 2);
102   G4LogicalVolume* preAbsorberLogical =
103       new G4LogicalVolume(preAbsorberSolid, G4Material::GetMaterial("G4_Cu"),
104                           "PreAbsorber");
105   new G4PVPlacement(nullptr, G4ThreeVector(0., 0., -absorberSizeZ / 2 -
106                                                    preAbsorberThickness / 2),
107                     preAbsorberLogical, "PreAbsorber", fpWorldLogical, false, 0);
108
109   // Fiber (Quartz)
110   G4double fiberRadius = 0.625 * mm;
111   G4double fiberLength = absorberSizeZ;
112   G4Tubs* fiberSolid =
113       new G4Tubs("Fiber", 0., fiberRadius, fiberLength / 2, 0., 360. * deg);
114   G4LogicalVolume* fiberLogical =
115       new G4LogicalVolume(fiberSolid, G4Material::GetMaterial("G4_SILICON_DIOXIDE"),
116                           "Fiber");
117
118   // PMMA Cladding
119   G4double claddingThickness = 0.1 * mm;
120   G4Tubs* claddingSolid = new G4Tubs(
121       "Cladding", fiberRadius, fiberRadius + claddingThickness, fiberLength / 2,
122       0., 360. * deg);
123   G4LogicalVolume* claddingLogical =
124       new G4LogicalVolume(claddingSolid, G4Material::GetMaterial("G4_PLEXIGLASS"),
125                           "Cladding");
126   new G4PVPlacement(nullptr, G4ThreeVector(), claddingLogical, "Cladding",
127                     fiberLogical, false, 0);
128
129   // Placement of fibers in a 3x3 grid
```

```
  G4double fiberPitch = 2.632 * mm;
  for (G4int i = -1; i <= 1; i++) {
    for (G4int j = -1; j <= 1; j++) {
      G4double xPosFiber = i * fiberPitch;
      G4double yPosFiber = j * fiberPitch;
      G4cout << "Fiber position X: " << xPosFiber << " mm, Y: " << yPosFiber
             << " mm" << G4endl;
      new G4PVPlacement(nullptr, G4ThreeVector(xPosFiber, yPosFiber, 0.),
                        fiberLogical, "Fiber", absorberLogical, false,
                        (i + 1) * 3 + (j + 1));
    }
  }

  // Visualization attributes
  fpWorldLogical->SetVisAttributes(G4VisAttributes::GetInvisible());

  G4VisAttributes* absorberVisAtt = new G4VisAttributes(G4Colour::Gray());
  absorberVisAtt->SetVisibility(true);
  absorberLogical->SetVisAttributes(absorberVisAtt);

  G4VisAttributes* preAbsorberVisAtt = new G4VisAttributes(G4Colour::Brown());
  preAbsorberVisAtt->SetVisibility(true);
  preAbsorberLogical->SetVisAttributes(preAbsorberVisAtt);

  G4VisAttributes* fiberVisAtt = new G4VisAttributes(G4Colour::Blue());
  fiberVisAtt->SetVisibility(true);
  fiberLogical->SetVisAttributes(fiberVisAtt);

  G4VisAttributes* claddingVisAtt = new G4VisAttributes(G4Colour::White());
  claddingVisAtt->SetVisibility(true);
  claddingLogical->SetVisAttributes(claddingVisAtt);
}
```

Listing A.1: MyDetectorConstruction.hh/cc

A.2 MyPhysicsList

```
#include "MyPhysicsList.hh"
#include "G4DecayPhysics.hh"
#include "G4EmStandardPhysics_option4.hh"
#include "G4HadronElasticPhysicsHP.hh"
#include "G4HadronPhysicsFTFP_BERT.hh"
#include "G4IonPhysics.hh"
#include "G4OpticalPhysics.hh"
#include "G4StoppingPhysics.hh"

MyPhysicsList::MyPhysicsList() : G4VModularPhysicsList() {
  SetVerboseLevel(1);

  // Default physics
  RegisterPhysics(new G4DecayPhysics());

  // EM physics
  RegisterPhysics(new G4EmStandardPhysics_option4());

  // Hadronic physics
  RegisterPhysics(new G4HadronPhysicsFTFP_BERT());
  RegisterPhysics(new G4HadronElasticPhysicsHP());
  RegisterPhysics(new G4StoppingPhysics());
  RegisterPhysics(new G4IonPhysics());

  // Optical physics
  G4OpticalPhysics* opticalPhysics = new G4OpticalPhysics();
  auto opticalParams = G4OpticalParameters::Instance();
  opticalParams->SetCerenkovMaxPhotonsPerStep(100);
  opticalParams->SetCerenkovTrackSecondariesFirst(true);
  RegisterPhysics(opticalPhysics);
}

MyPhysicsList::~MyPhysicsList() {}

void MyPhysicsList::SetCuts() {
  G4VUserPhysicsList::SetCuts();
}
```

Listing A.2: MyPhysicsList.hh/cc

A.3 MyPrimaryGeneratorAction

```
#include "MyPrimaryGeneratorAction.hh"

#include "G4Event.hh"
#include "G4ParticleGun.hh"
#include "G4ParticleTable.hh"
#include "G4SystemOfUnits.hh"

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

  // Default gun settings - gamma particle with an energy of 100 MeV
  G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
  G4ParticleDefinition* particle = particleTable->FindParticle("gamma");

  particleGun->SetParticleDefinition(particle);
  particleGun->SetParticleMomentumDirection(G4ThreeVector(0., 0., 1.));
  particleGun->SetParticleEnergy(100. * MeV);
  particleGun->SetParticlePosition(G4ThreeVector(0., 0., -15. * cm));
}

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

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

Listing A.3: MyPrimaryGeneratorAction.hh/cc

A.4 MyRunAction

```
#include "MyRunAction.hh"

#include "G4Run.hh"
#include "G4SystemOfUnits.hh"
#include "MyAnalysis.hh"

MyRunAction::MyRunAction() {
  G4cout << "MyRunAction initialized!" << G4endl;
}

MyRunAction::~MyRunAction() {}

void MyRunAction::BeginOfRunAction(const G4Run*) {
  // Get analysis manager
  auto analysisManager = G4AnalysisManager::Instance();

  // Create ntuple
  analysisManager->CreateNtuple("Photons", "Photon Data");
  analysisManager->CreateNtupleIColumn("EventID");
  analysisManager->CreateNtupleDColumn("Energy");
  analysisManager->CreateNtupleDColumn("Time");
  analysisManager->FinishNtuple();

  // Open an output file
  analysisManager->OpenFile("MyOutput.root");
}

void MyRunAction::EndOfRunAction(const G4Run*) {
  // Get analysis manager
  auto analysisManager = G4AnalysisManager::Instance();
  if (!analysisManager) {
    G4cerr << "ERROR: AnalysisManager is null in EndOfRunAction!" << G4endl;
    return;
  }

  // Save histograms & ntuple and close file
  analysisManager->Write();
  analysisManager->CloseFile();
  G4cout << "Dane zosta y zapisane do pliku MyOutput.root" << G4endl;
}
```

Listing A.4: MyRunAction.hh/cc

A.5 MySteppingAction

```cpp
#include "MySteppingAction.hh"

#include "G4OpticalPhoton.hh"
#include "G4SteppingManager.hh"
#include "G4SystemOfUnits.hh"
#include "G4Track.hh"
#include "MyAnalysis.hh"
#include "G4EventManager.hh"
#include "G4SystemOfUnits.hh"
#include "G4Cerenkov.hh"
#include "G4StepPoint.hh"
#include "G4ProcessManager.hh"

MySteppingAction::MySteppingAction(): totalEnergyDeposit(0.) {}

MySteppingAction::~MySteppingAction() {}

void MySteppingAction::UserSteppingAction(const G4Step* step) {

  // G4cout << "UserSteppingAction executed!" << G4endl;
  G4LogicalVolume* volume = step->GetPreStepPoint()->GetTouchableHandle()->GetVolume()->
    GetLogicalVolume();
  if (volume->GetName() == "Fiber") {
    // Get the energy deposited in this step
    G4double edep = step->GetTotalEnergyDeposit();

    // Add the energy deposit to the total
    totalEnergyDeposit += edep;
  }

  // Check if the particle is an optical photon
  G4Track* track = step->GetTrack();
  // G4cout << "Cz stka: " << track->GetDefinition()->GetParticleName() << G4endl;
  // G4cout << "Cz stka: " << G4OpticalPhoton::OpticalPhotonDefinition() << G4endl;
  // if (track->GetDefinition() == G4OpticalPhoton::OpticalPhotonDefinition()) {
  //   G4cout << "Optical photon detected!" << G4endl;
  // }

  // G4cout << (track->GetDefinition()->GetParticleName() != "opticalphoton") << G4endl;
  if (track->GetDefinition()->GetParticleName() != "opticalphoton")
    return;

  // Check if the photon reaches the end of the fiber
  G4StepPoint* postStepPoint = step->GetPostStepPoint();
  G4double fiberLength = 50.0 * cm;

  // G4cout << (postStepPoint->GetPosition().z() < (fiberLength / 2)) << G4endl;
  // if (postStepPoint->GetPosition().z() < fiberLength / 2) return;

  // G4ProcessManager* pmanager = track->GetDefinition()->GetProcessManager();
  // const G4Cerenkov* cerenkovProcess =
  //     dynamic_cast<const G4Cerenkov*>(pmanager->GetProcess("Cerenkov"));
  // G4cout << (!cerenkovProcess || step->GetPostStepPoint()->GetProcessDefinedStep() !=
    cerenkovProcess) << G4endl;
  // if (!cerenkovProcess || step->GetPostStepPoint()->GetProcessDefinedStep() !=
    cerenkovProcess) return;

  // Get a pointer to the analysis manager
  auto analysisManager = G4AnalysisManager::Instance();

  // Get the event number
  G4int eventID =
      G4EventManager::GetEventManager()->GetConstCurrentEvent()->GetEventID();

  // Get the photon's energy and time
  G4double energy = track->GetKineticEnergy();
  G4double time = track->GetGlobalTime();

  // G4cout << "Photon detected! EventID: " << eventID
  //        << ", Energy: " << energy / MeV << " MeV, Time: " << time / ns
  //        << " ns" << G4endl;

  // Save the data to the ntuple
  analysisManager->FillNtupleIColumn(0, eventID);
```

```
  analysisManager->FillNtupleDColumn(1, energy / MeV);
  analysisManager->FillNtupleDColumn(2, time / ns);
  analysisManager->AddNtupleRow();
}
```

Listing A.5: MySteppingAction.hh/cc

A.6 Makro analizy ROOT

```
// analyzePhotons.C
//
// Macro to analyze an ntuple containing optical photon data.
// It automatically adjusts the histogram ranges to fit the data,
// then creates and draws histograms for the energy (in MeV) and time (in ns)
// of optical photons reaching the end of the fiber.

void analyzePhotons() {
  // Open the ROOT file containing the ntuple
  TFile* file = TFile::Open("MyOutput.root");
  if (!file || file->IsZombie()) {
    std::cout << "Error: Could not open file MyOutput.root" << std::endl;
    return;
  }

  // Get the TTree "Photons" from the file
  TTree* tree = (TTree*)file->Get("Photons");
  if (!tree) {
    std::cout << "Error: TTree 'Photons' not found in the file." << std::endl;
    return;
  }

  // Set up variables to read data from the tree branches
  Double_t energy = 0;
  Double_t time = 0;
  tree->SetBranchAddress("Energy", &energy);
  tree->SetBranchAddress("Time", &time);

  // Automatically determine the range for the energy histogram
  Double_t minEnergy = tree->GetMinimum("Energy");
  Double_t maxEnergy = tree->GetMaximum("Energy");
  if (minEnergy == maxEnergy) {
    // If all data points have the same energy, add a default margin
    minEnergy -= 1;
    maxEnergy += 1;
  } else {
    // Add a 10% margin to both sides
    Double_t marginE = 0.1 * (maxEnergy - minEnergy);
    minEnergy -= marginE;
    maxEnergy += marginE;
  }

  // Automatically determine the range for the time histogram
  Double_t minTime = tree->GetMinimum("Time");
  Double_t maxTime = tree->GetMaximum("Time");
  if (minTime == maxTime) {
    minTime -= 1;
    maxTime += 1;
  } else {
    Double_t marginT = 0.1 * (maxTime - minTime);
    minTime -= marginT;
    maxTime += marginT;
  }

  // Create a histogram for the energy of optical photons
  TH1D* hEnergy = new TH1D("hEnergy", "Optical Photons Energy Distribution", 100,
    minEnergy, maxEnergy);
  hEnergy->GetXaxis()->SetTitle("Energy (MeV)");
  hEnergy->GetYaxis()->SetTitle("Number of Photons");

  // Create a histogram for the time of optical photons
  TH1D* hTime = new TH1D("hTime", "Optical Photons Time Distribution", 100, minTime,
    maxTime);
  hTime->GetXaxis()->SetTitle("Time (ns)");
  hTime->GetYaxis()->SetTitle("Number of Photons");

  // Fill the histograms with data from the tree
```

```
  Long64_t nEntries = tree->GetEntries();
  for (Long64_t i = 0; i < nEntries; ++i) {
    tree->GetEntry(i);
    hEnergy->Fill(energy);
    hTime->Fill(time);
  }

  // Create a canvas and divide it into two pads for drawing both histograms
  TCanvas* c = new TCanvas("c", "Optical Photons Histograms", 1200, 600);
  c->Divide(2, 1);

  // Draw the energy histogram in the first pad
  c->cd(1);
  hEnergy->Draw();

  // Draw the time histogram in the second pad
  c->cd(2);
  hTime->Draw();

  // Save the canvas as a PNG image
  c->SaveAs("PhotonHistograms.png");
}
```

Listing A.6: analyzePhotons.C