smsimulator

1. Project Goals

This project aims to perform detailed particle physics simulations using `smsimulator` (based on Geant4) to achieve the following core objectives:

- **Optimization of Detector Layout**: To evaluate and optimize the relative positions of the Target and the Particle Drift Chambers (PDC) through simulation data, aiming for the best particle detection efficiency and resolution.
- **Full-chain Simulation and Reconstruction**: To establish a complete simulation workflow, from particle generation, transport, and detection to final data reconstruction and analysis, providing reliable data for physics analysis.

---

2. Simulation Stage (smsimulator)

2.1. Simulation Control and Data Output

We use Geant4 macro files (`.mac`) to flexibly control various parameters in the simulation, especially the positions of the detectors and the target. This allows for a systematic scan of different geometric layouts.

After the simulation, key physics information is saved in a ROOT Tree. It primarily contains two types of data:

- **Incident Particle Information**:
    - `n`: Neutron information in the event.
    - `p`: Proton information in the event.
    - `momente`: Momentum vector of the particles.
- **Detector Hit Information**:
    - **PDC (Drift Chamber)**: Stored in the `FragSimData` branch, which records fragment information as particles pass through the PDC.
    - **NEBULA (Neutron Detector)**: Stored using a specific data format built into `smsimulator`.

2.2. Correction of Data Recording in `FragmentSD.cc`

In earlier versions of the simulation code, we discovered that `FragmentSD.cc` (the sensitive detector class for handling fragments) did not correctly record the energy deposit of particles in the detector. To fix this, we modified the code as follows to ensure the energy deposit is accurately calculated and stored:

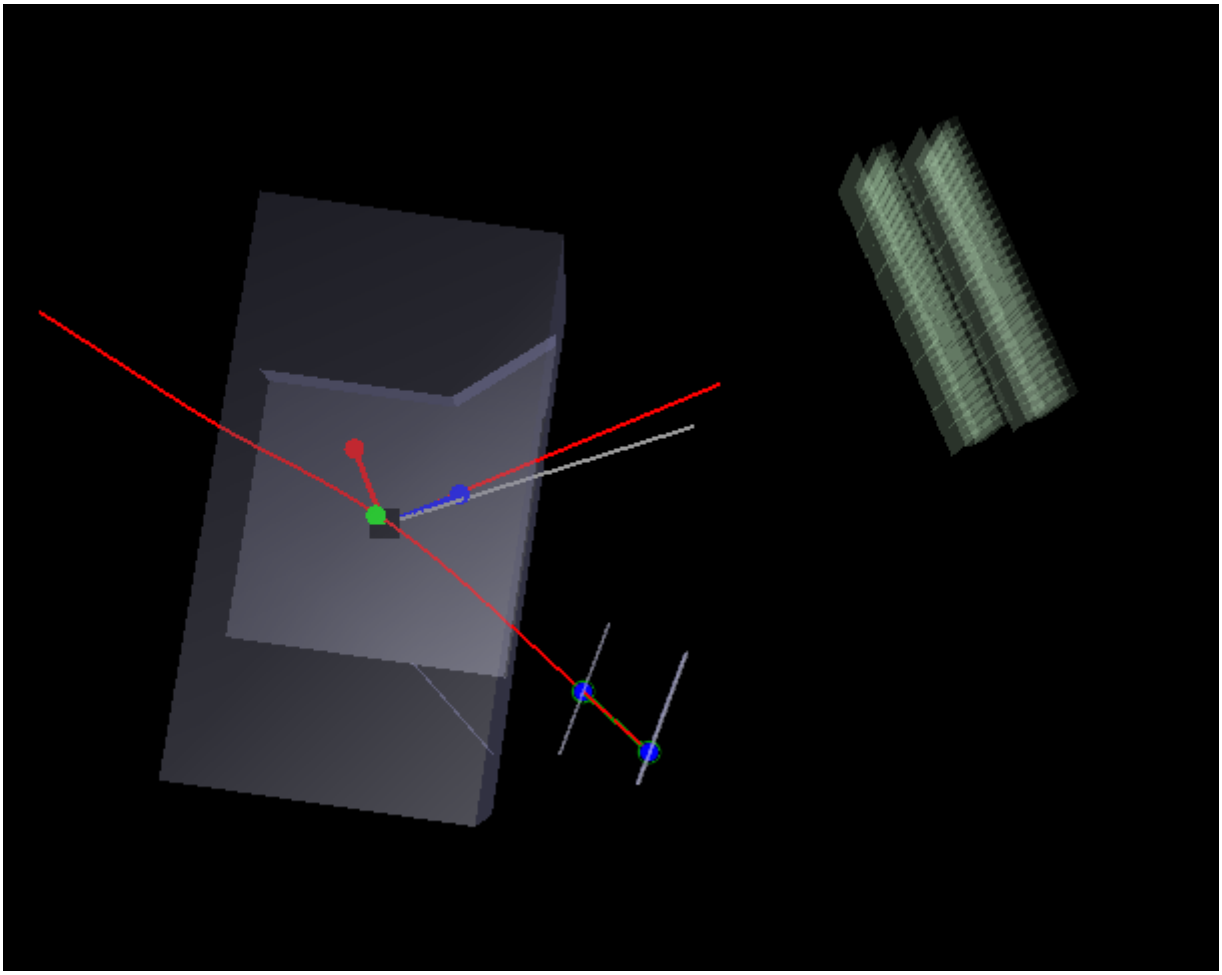
```
// Get the SimDataManager instance
SimDataManager *sman = SimDataManager::GetSimDataManager();
// Find the data array named "FragSimData"
TClonesArray *SimDataArray = sman->FindSimDataArray("FragSimData");

// ... inside the loop processing each step ...
```

```
// Get the total energy deposit from aStep and convert it to MeV
Double_t energyDeposit_MeV = aStep->GetTotalEnergyDeposit() / MeV;

// ...

// Store the calculated energy deposit into the data object
data->fEnergyDeposit = energyDeposit_MeV;
```

This correction ensures that the energy deposit information can be accurately used in subsequent analysis.

---

3. Analysis and Reconstruction Stage



To process and analyze the data generated by `smsimulator`, we have developed a ROOT-based analysis tool library. The structure of this library is as follows:

```
.
├── macros/                    # ROOT analysis macros
│   ├── run_display.C          # Main display script for event
visualization
│   ├── test_run_display.C     # Functional test script
│   └── ...
├── sources/                   # C++ source code
│   ├── include/
│   │   ├── EventDataReader.hh    # Event data reading
```

```
│   │   ├── EventDisplay.hh       # 3D event display
│   │   ├── GeometryManager.hh    # Geometry parameter management
│   │   ├── MagneticField.hh      # Magnetic field data handling
│   │   ├── ParticleTrajectory.hh # Particle trajectory calculation
│   │   ├── PDCSimAna.hh          # PDC data reconstruction
│   │   ├── RecoEvent.hh          # Data structure for reconstructed events
│   │   └── TargetReconstructor.hh# Target momentum reconstruction
│   └── src/
│       ├── ... (corresponding .cc implementation files)
└── magnetic_field_test.root    # Pre-processed magnetic field data
```

3.1. Core Analysis Tools Explained

3.1.1. `GeometryManager`

This class is responsible for loading and managing the experimental geometry from external files. During initialization, it sets default geometric parameters, such as the positions and angles of the PDCs, which can be updated from a configuration file.

```
// GeometryManager constructor
GeometryManager::GeometryManager()
    : fAngleRad(0.0), fPDC1_Position(0,0,0), fPDC2_Position(0,0,0)
{
    // Default values, can be updated later by LoadGeometry
}
```

3.1.2. `ParticleTrajectory`

This is the core class for trajectory calculation. It uses the **4th-order Runge-Kutta integration method** to accurately calculate the trajectory of charged particles in a non-uniform magnetic field. This method significantly improves the accuracy of the trajectory calculation by performing four intermediate evaluations within each time step `dt`.

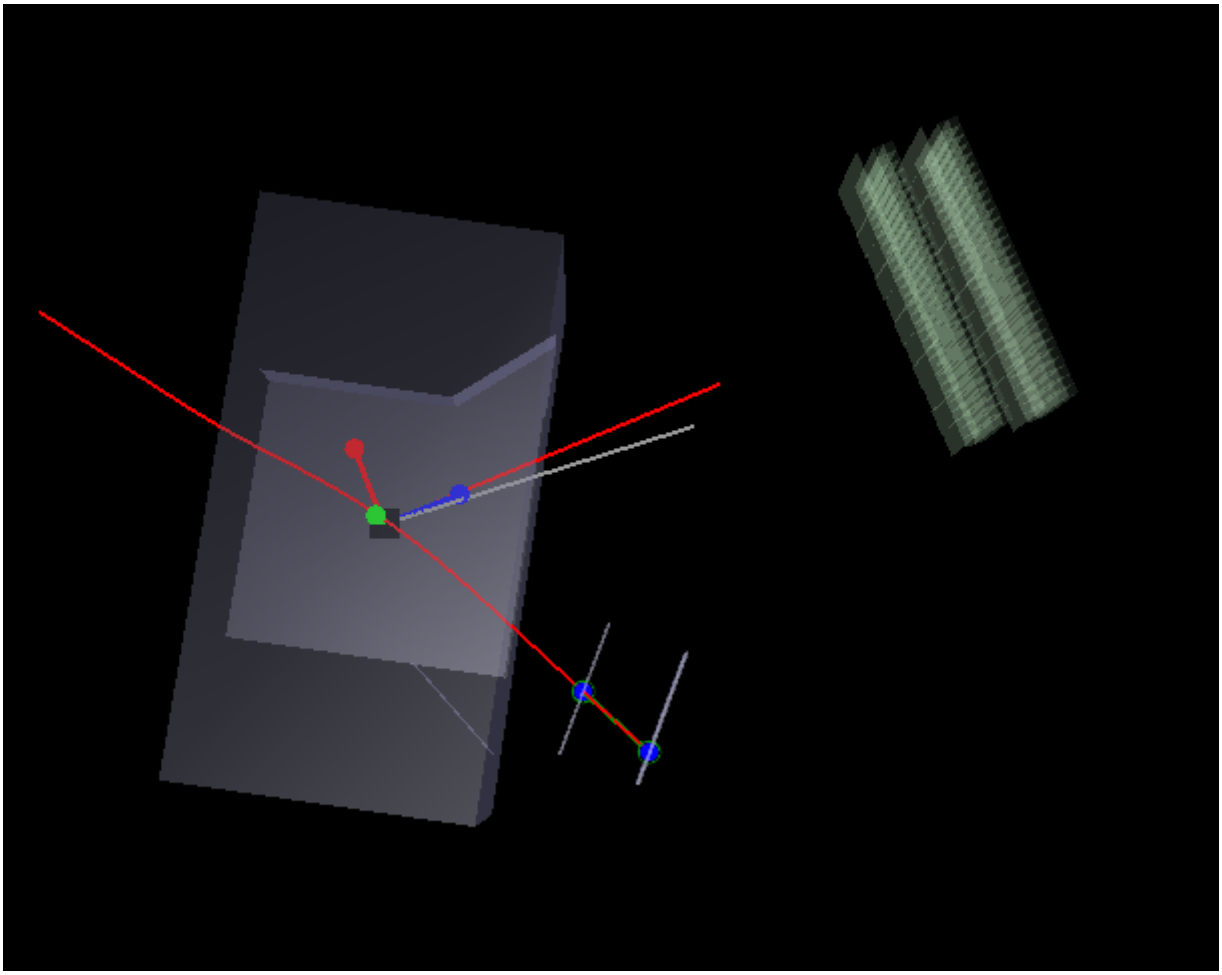
The core implementation of the Runge-Kutta algorithm in the code is shown below:

```
// Calculate initial energy
double E0 = TMath::Sqrt(p0.Mag2() + mass*mass);

// --- 4th-Order Runge-Kutta Integration Steps ---
// K1: Derivatives at time t0
TVector3 k1_r = p0 * (kSpeedOfLight * kSpeedOfLight / E0); // dr/dt = v
TVector3 k1_p = CalculateForce(r0, p0, charge);           // dp/dt = F

// K2: Derivatives at time t0 + dt/2
TVector3 r1 = r0 + k1_r * (dt/2);
TVector3 p1 = p0 + k1_p * (dt/2);
// ... calculate k2_r and k2_p ...
```

```
// K3: Derivatives at time t0 + dt/2 (second estimate)
// ...

// K4: Derivatives at time t0 + dt
// ...

// --- Final Step Update ---
// Weighted average update for position and momentum
TVector3 r_new = r0 + (k1_r + 2*k2_r + 2*k3_r + k4_r) * (dt/6);
TVector3 p_new = p0 + (k1_p + 2*k2_p + 2*k3_p + k4_p) * (dt/6);
```

3.2. Visualization Results

We use the `EventDisplay` class to visualize the simulation and reconstruction results in 3D. The image below shows a typical event display, including the detector geometry, particle tracks, and reconstructed points.



---

4. Current Issues and To-Do List

In our current analysis, we have identified the following issues that need to be addressed:

- **Incorrect Target Position**: The position of the target may not be set correctly during the input stage for `(n, p)` events, leading to an inaccurate starting point for reconstruction.

- **Initial Particle Rotation**: The initial neutrons and protons need to be rotated by **-5 degrees** around the Y-axis at generation to match the coordinate system of the SAMURAI magnet.

The next steps will focus on resolving these issues and further refining the reconstruction algorithms to improve momentum and position resolution.

now