Overall class structure in Geant4

Geant4 Training Course in Medicine 2023

Einar Elén

September, 2023

1. Introduction

1.1. Credits

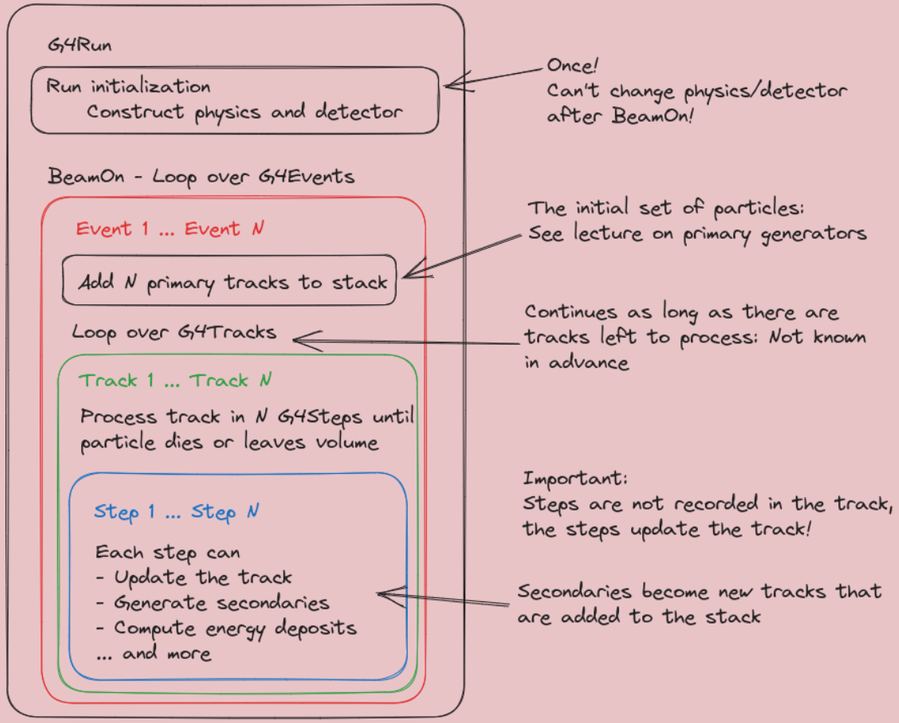
This lecture structure is heavily based on a previous lecture by Ivana Hrivnacova and similar lectures in the past (but condensed).

1.2. How is Geant4 structured?

- Geant4 is not an application
  - Geant4 is a library that provides you with building blocks you need
  - i.e. It is a toolkit, and a very flexible one at that!
- Use the parts you need for your problem
  - Upside: Allows Geant4 to work for very small scale simple problems up to huge ones
  - Downside: Some assembly required…
- A lot of information can be found in the Book for application developers https://geant4.web.cern.ch/docs/

2. Kernel classes

- Kernel classes in Geant4 important concepts corresponding to different phases of the simulation
- Represented by classes and a corresponding manager class
- We will go from the largest to the smallest
  - Run
  - Event
  - Track
  - Step and StepPoint
- Note: Many Geant4 classes can be default or user derived versions, I won't repeat this here



2.1. Run

- Run is a configuration and a set of events
- Before a run starts, user must have configured
  - Detector setup
  - Source setup
  - Physics processes
- After configuration and initialization typically consists of a single *event-loop*
  - Simulation starts by specifying “BeamOn” command, analogous to experiments
    - Here relevant physics configuration, cross-section table calculations, and geometry optimization occurs
  - Configurations must not be changed until a run is over!
- Represented by `G4Run`, managed by the `G4RunManager`

2.2. Event

- The basic unit of simulation in Geant4
  - Other simulation tools may use different names for the same concept (e.g. “history”)
- Start with primary particles/tracks in a stack
- Pop off one particle at a time and track it as it propagates through the geometry until it is done
  - Along the way secondary particles can be produced and are then added to the stack
  - Event is over when stack is empty
- Represented by `G4Event`, managed by `G4EventManager`

2.3. Track

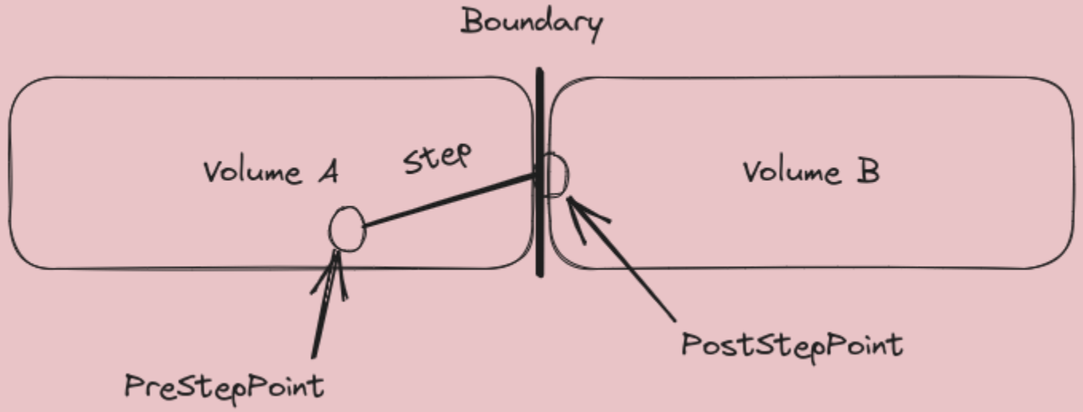
- A track is a *snapshot* of a particle
  - Current physical quantities
    - Position, time, energy, etc
    - *in current part of the simulation*
  - No record of previous quantities
- Common misconception is that a track is a history of changes!
  - Updated through series of steps, but does not consist of them
- Represented by `G4Track`, managed by `G4TrackingManager`

2.3.1. Track lifetime

- Track keeps propagating until the particle
  - Leaves the outermost volume,
  - Disappears during an interaction,
  - It has zero kinetic energy *and* and has no `AtRest` process, or
  - It is killed artifically by the user
- Remember: Event continues until all tracks are gone
  - No tracks persist at the end of the event!
  - If you need this information, use the `G4Trajectory` class

2.4. Step

- Represents the delta (change) of a particle to be applied to a track
  - Two points: Pre/Post StepPoint
- Step is represented by `G4Step` while the pre/post step points are represented by `G4StepPoint`
- Both are managed by `G4SteppingManager`



- When a step is limited by a volume boundary, the endpoint is at the boundary but *belongs to the next volume*
  - You can therefore simulate boundary processes if needed

3. Particles and tracking

3.1. Particles

- How particles are represented can sometimes be confusing for beginners
  - Not a single particle class
  - Different aspects of a particle are represented by different types
- `G4Track`, `G4DynamicParticle`, and `G4ParticleDefinition`

3.2. The three particle representations

- `G4Track`
  - A snapshot of a particle currently being tracked
- `G4DynamicParticle`
  - Represents the dynamic physical properties of an *individual* particle
  - Momentum, energy, spin etc
  - Each `G4Track` has its *own and unique* `G4DynamicParticle`
- `G4ParticleDefinition`
  - Represents the static properties of a particle
  - Charge, mass, lifetime etc
    - `G4ParticleDefinition` is shared between all `G4DynamicParticle` objects of the same type
  - Also stores the list of physics processes involving the particle

3.3. Tracking and processes

- Tracking in Geant4 is generic
  - Works the same way independent of the particle type
  - This is powerful, but can come at a performance cost
- Conceptually the algorithm is straight-forward
- For more details, see the documentation

3.3.1. Tracking loop

- Obtain the list of physics processes from each particle type
- In turn let each process (if applicable)
  - Contribute to determining step length
  - Contribute to changes of the physical quantities of the track
  - Production of secondary particles
  - Suggest changes in track state (e.g. killing it)

4. User applications

- Geant4 also provides classes that help you build your application by interacting with the kernel classes.
  - Some are mandatory
  - Some are optional
  - All are passed to the `G4RunManager`

4.1. What do you need to do?

- Implement a `main()` function
- Create derived versions of the mandatory classes as well as any optional ones you need
- Construct a `G4RunManager` and set it up with them
- Handle anything else your application needs, either with Geant4 or other means!
  - GUI, CLI, make histograms, analysis, etc

4.1.1. What should you not do?

- In your application do not use the raw `std::cout` and `std::cerr` objects, use the `G4cout` and `G4cerr` versions
  - These will work correctly in all parts of Geant4
- Similarly: Avoid relying on raw `std::cin`, consider using the user-defined commands instead!

4.2. Mandatory classes

- `G4VUserDetectorConstruction` How the geometry is set up (volumes, materials etc)
- `G4VUserPhysicsList` The physics configuration
- `G4VUserActionInitialization` Action classes called during event processing
  - `G4VPrimaryGeneratorAction` is mandatory

Note: The `V` in the class name indicates that the class is abstract: You need to derive your own class from it

4.3. UserActionInitialization

- `G4VUserActionInitialization` groups and initializes all user action classes.
  - You override the `Build()` function where you instantiate any classes you want
  - Has to be used for multithreaded processing
  - Only the Primary generator action is mandatory (needed to produce the initial tracks)
- Action classes are called automatically at relevant phase of the simulation by the kernel

4.3.1. Optional classes

- Derive from the following and override the member functions you need
  - `G4UserRunAction`
  - `G4UserEventAction`
  - `G4UserTrackingAction`
  - `G4UserSteppingAction`
  - `G4UserStackingAction`
- For example: `G4UserTrackingAction::PreUserTrackingAction`
  - Called before tracking
- Since these are not abstract, make sure you are actually overriding!

5. Questions?


LUND
UNIVERSITY
GEANT4
A SIMULATION TOOLKIT