Geant4 HEPCAT Lab

Geant4 (GEometry ANd Tracking) is a simulation toolkit that allows the user to simulate particle interactions within a detector medium. It is an invaluable software when it comes to detector design, shielding design, and the interpretation of experimental data. The main goals of this lab are to:

- Learn how G4 is structured
- Set up a liquid xenon detector in Geant4 with photosensors
- Analyze the output of an optical simulation
- **Bonus:** simulate neutron scattering from Cf252

Geant4 Structure

Geant4 has three essential managers:

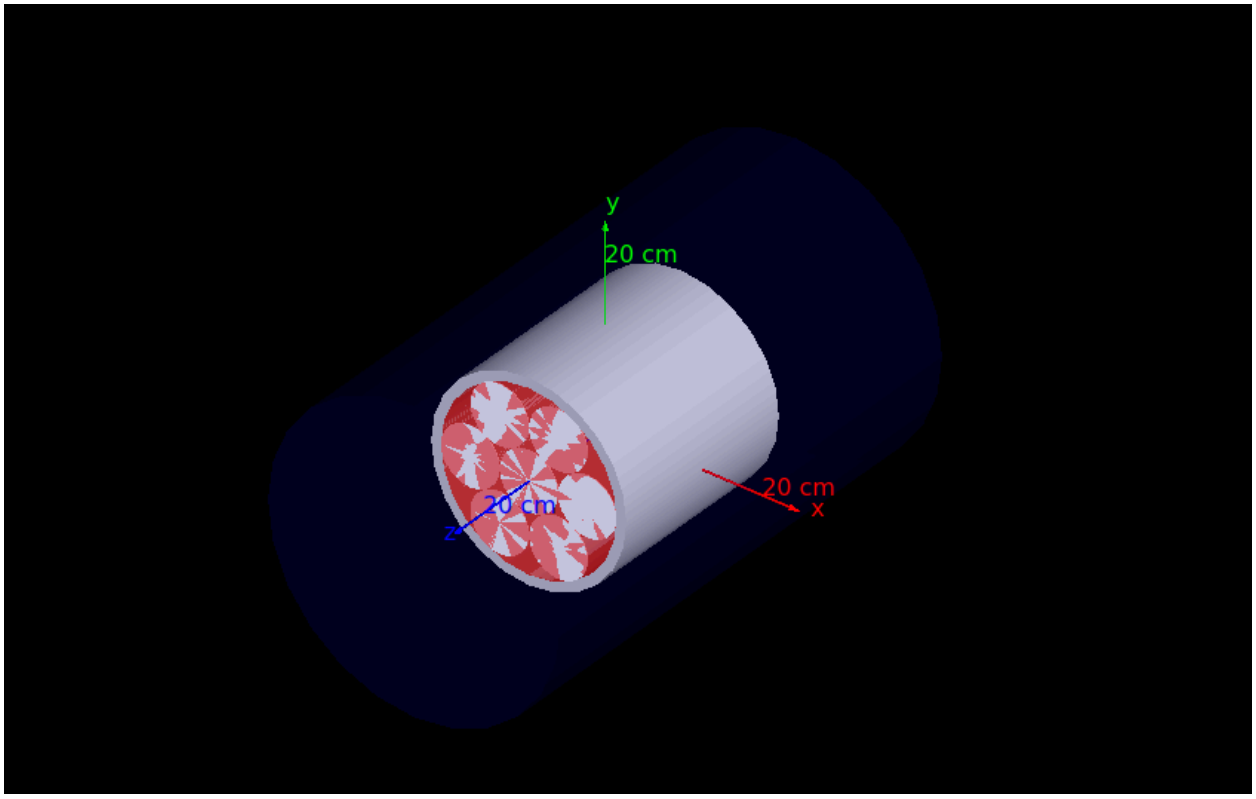
- Run manager:
    - In charge of the geometry, physics, and simulation. The “computer”
- Visualization manager:
    - In charge of graphics/visualization
- UI manager:
    - In charge of command passing

Of these, the basic inputs needed to start a program are:

- A detector construction:
    - The geometry and materials of the detector
    - Also includes sensitive detectors used to track particle interactions
- A physics list:
    - Provides processes for the particles to interact via
    - In principle there’s a lot to say, but it’s beyond the scope of this lab
- A generator:
    - Generates initial particles with a particular position, momentum, charge, polarization (if applicable), and type
- A run action (Absolutely necessary for multithreading only)
    - More on actions later

Detector construction:

The detector we will construct will be a liquid xenon detector with 14 PMTs in total (7 on top, 7 on bottom). And it will look like this:



The blue represents a liquid xenon (LXe) bath. The white barrel is a PTFE reflector of xenon light, and the red volume represents the inner LXe volume that will be used to track particle interactions.

All G4 volumes consist of:

- A solid volume
    - *Only* defines the shape
- A logical volume
    - Defines the material
    - Has subvolumes (i.e. can be a “mother” volume”)
    - Can write to the color
- Physical volume:
    - Defines the position, rotation, and its placement in a mother volume

Exercise:

Fill in the comments within HepcatConstruction.cc to construct the geometry. The dimensions of the geometry are given in HepcatConstruction.hh. Geometry building is the most straightforward part of Geant4 (in my opinion) and is well documented in the user guide for application developers. (See section 4.1 and always feel free to ask your TA for help!)

In the `Construct()` *method:*

1. Make a LXe cylinder volume whose mother volume is the world with the following attributes:
    a. `dLXeRadius` as the radius
    b. `dLXeHeight` as the height
    c. Mother volume is world
    d. Name this `solid_lxe_volume`, `logic_lxe_volume`, and `phys_lxe_volume`
2. Make a PTFE shell with:
    a. `dShellInnerRadius` inner radius
    b. `dShellThickness` thickness
    c. `dShellHeight` height
    d. Mother volume is LXe
    e. Name this `solid_ptfe`, `logic_ptfe`, `phys_ptfe`
3. An inner LXe volume
    a. This is going to be a **sensitive detector**, which is why it's a redundant liquid xenon volume that overlaps with the main LXe volume
    b. This is a cylinder that should fully cover the inside of the PTFE shell
    c. Name this `solid_lxe_inner`, `logic_lxe_inner`, `phys_lxe_inner`
4. 14 PMT windows, 7 on top, 7 on bottom
    a. The height of these windows should be the same as `dPmtWindowThickness`
    b. Radius is `dPmtRadius`
    c. Hint: Use translation vectors to place the PMTs, there should only be one logical volume but multiple physical volume placements
    d. Utilize the copy number in the physical volume placement to number the PMTs
    e. Name these `solid_pmt_window`, `logic_pmt_window`, and `phys_pmt_window`

Run `./sdsim -v true` once you think you've made something!

Particle Source, Generator, and Actions:

The largest unit of a simulation is called a ***run***, which creates multiple ***events***. An event is just the initialization of a bunch of particles specified positions, momenta, polarizations (if applicable), and particle types. A run with n events is started with the following command:

```
/run/beamOn [n]
```

Now is a good time to discuss what an *action* is. The run manager takes on multiple *actions*, which are just "things for the program to do" either at the beginning and/or end of a unit of simulation. For example, the `HepcatRunAction` will open up a data file to write data to at the `BeginOfRunAction`, and write and close that file at the `EndOfRunAction`. Likewise, the run manager calls on `GeneratePrimaries` within the ***generator action*** (in this case, `HepcatGeneratorAction`) to start making events. Finally, at the end of an event, once all particles and secondary particles have either been absorbed or exited the world, the `HepcatEventAction::EndOfEventAction` will fill the output file with the data of the event (more on what this means when we talk about sensitive detectors).

The `GeneratePrimaries` method of `HepcatGeneratorAction` calls on a `G4VPrimaryGenerator` object's (usually called a particle source) `GeneratePrimaryVertex` method.

Exercise: In this exercise, you will fill in the details for how to generate particles with an isotropic momentum with an azimuthal angle between `m_dMinPhi` and `m_dMaxPhi`, and a polar angle between `m_dMinTheta` and `m_dMaxTheta`.

Hints:

1. G4UniformRand() gets you a random number between 0 and 1
2. Think about how the differential for solid angle is $\sin\theta d\theta d\varphi = d(\cos\theta)d\varphi$

Sensitive Detectors:

Sensitive detectors are the *active* geometry components tied to ***logical*** volumes. They are associated with *hits*, which are interactions of particles within the logical volume. Sensitive detectors are equipped with a method called:

```
ProcessHits(G4Step *pStep, G4TouchableHistory *pHistory)
```

which is automatically called during the simulation (more on the specifics later). These hits then get filled into a *hits collection* which can be accessed at the end of an *event*.

Exercise: There isn't much to do here. Sensitive detectors and hits are beyond the scope of what we can do for now. And besides, everyone just copies and pastes each others' code anyway.

1. The `ConstructSDandField()` method of the detector construction is *necessary* for multithreading. You do your sensitive detector settings in here
2. Uncomment those lines

3. *Question:* Why is it necessary to have some volumes saved as class members?

Running the Program:

At this point, you should have enough to try some stuff out! A visualization macro (instructions file) is already written for you, though you may have to change around some of the variable names.

Exercise:

1. `./sdsim -v true` to run the program.
2. Try generating some photons:
    a. `/source/particleType opticalphoton`
    b. `/source/numParticles`
    c. Press the green play button

Analysis:

Exercise:

1. Create a macro file (file of commands) which contains the lines:
    a. `/source/particleType opticalphoton`
    b. `/source/numParticles [as many photons as you want to generate per event]`
    c. `/run/beamOn [number of events you want to simulate]`
2. Analyze the output root file in whichever way you like, ROOT, uproot (python), etc.
    a. Calculate the light collection efficiency in different parts of the detector, see if the hitpattern of the light on the PMTs tells you anything about the particle position
3. If time/curiosity permits, try generating other particles!