The Geant4 toolkit and basic user workflow

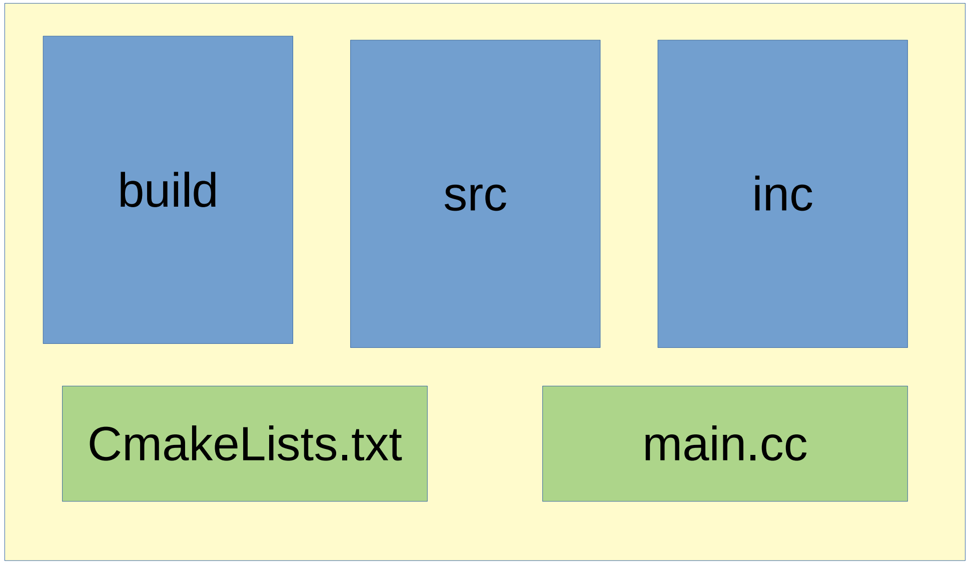
Geant4 is the name given to a collection of files (header files, source files, databases files…) that the user puts on his computer, and is supposed to use as a toolbox when he writes a C++ program to simulate particle transport.

Normal usage:

The usual way that this is done *by the user* is with the following file structure for a simulation:

- In the main project folder...
  - A "main.cc" source file
  - A "CmakeLists.txt" file
  - A "build" folder
  - A "src" folder
  - An "inc" folder

Project folder:



Some details on user files and folders

- The “**src**” directory contains all (except one, “main.cc”) of the C++ source code files for the simulation. The customary extension is “.cc”.
- The “**inc**” directory contains all the C++ header files for the simulation. The customary extension is “.hh”.
- The “**main.cc**” file is the top-level source code for the user’s simulation.
- The “**CmakeLists.txt**” is the file that tells the compiler where all the user’s source and header files are (in “src” and “inc”). It also does some compilation magic that we do not understand.
- The “**build**” folder will contain the executable for the simulation. This directory can also contain “macro” files, which are scripts that can be used to control a simulation. Geant4 is able to parse these text files, and the user does not need to directly modifying C++ code. The “main” executable is usually created in the following way:
  1) Change to the “build” directory (“cd build”).
  2) Run cmake (“cmake ../ -DGeant4_DIR=/path/to/geant4/install”, with the “../” for where the “CmakeLists.txt” is).
  3) Run make (“make”).
  4) Run the application (for example “./main” or “./main myMacro.mac”).

The Geant4 object-oriented paradigm

- The purpose of Geant4 is to simulate particle transport through materials, taking into account any physical processes the user wishes to include.
- *Everything in Geant4 is an object:* particles, steps, trajectories, processes, etc.
- Geant4 makes use of C++ objects (classes):
  - An object **can have** associated variables and functions.
    - Example: Object “Car” has variable “maxSpeed” and function “TurnOn()”.
  - An object **can inherit** variables and functions from a (possibly abstract) class higher in a hierarchy.
    - Example: Object “Car” inherits from the abstract class “Vehicle” which has the same variables “maxSpeed” and function “TurnOn()”.
  - An object **can extend** the capabilities of the class from which it inherits its properties.
    - Example: Object “Car” has a function “OpenSunroof()”, which object “Submarine” (also derived from “Vehicle”) does not have.

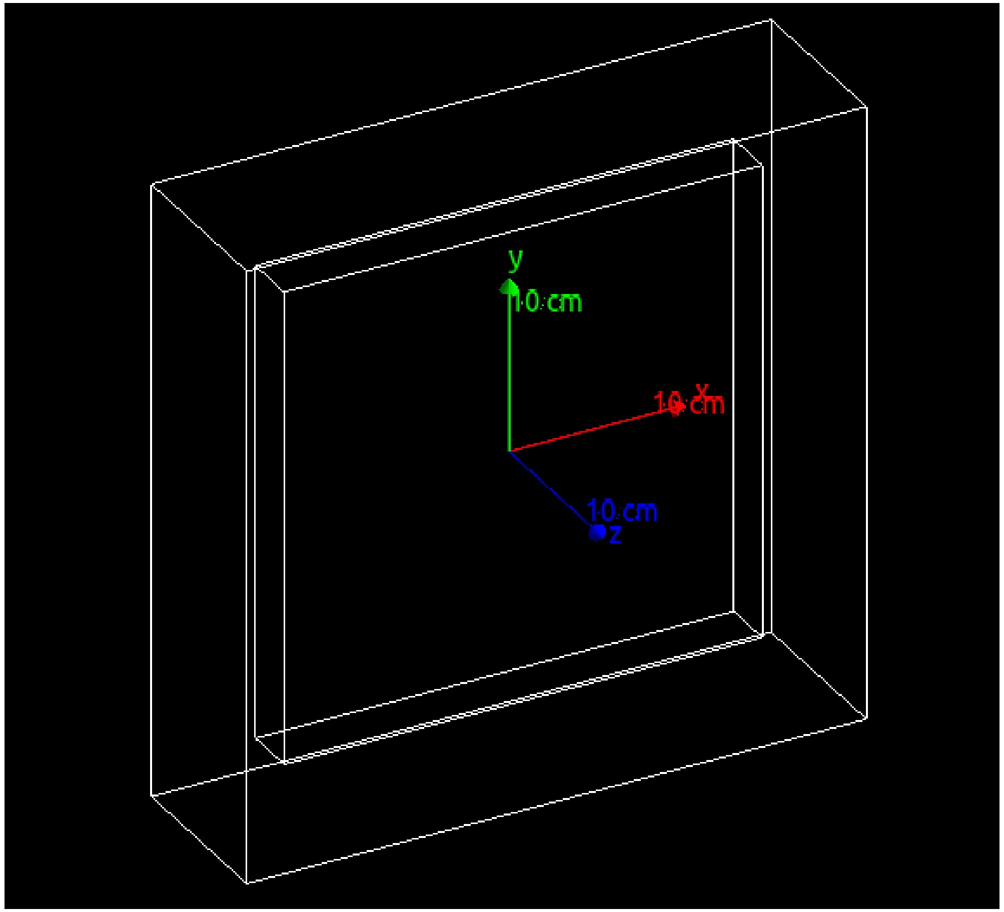
The required objects of a “bare-bones” Geant4 simulation

- Geant4 provides some classes we must put in our “main.cc” without any modification:
  - A “Run Manager” class.
  - A “User Interface Manager” class.
  - A “Visualization Manager” class.
- However, all Geant4 simulations must have four basic **user-created** objects:
  1) A “Detector” object.
     - Contains all geometry and material information.
  2) A “Physics List” object.
     - Contains all particles and physics processes which are allowed.
  3) A “Primary (Particle) Generator Action” object.
     - Can generate the primary particle(s) the user wishes to add to simulation.
  4) An “Action Initialization” object.
     - Puts the “Primary Generator Action” object into use.
     - Can put other user actions into use.
- **Such a Geant4 will not be of any use! We must ask the simulation for information.**
  - The internal machinery of the Geant4 toolkit will happily go ahead and calculate the transport of the particles we asked for through the materials that we created using the physics we desire. The only problem with that none of this information is saved anywhere by default. We must request to output it somewhere.

How do we get our simulation to output data? (Answer: User Action Classes)

- First we must explain the hierarchy of particle transport:
  - The “**Track**” object is the lowest level class in particle transport. This object has information about a particle at a given instant in time.
  - The next lowest level object is the “**Step**”. This has information about what has changed between two tracks (points).
  - The “**Event**” object consists of a collection of tracks that are propagated through the simulation. The tracks live on a stack, which is pulled from until empty.
  - The “**Run**” object is the highest level, and consists of a collection of events.
- Each of these particle transport objects (Track, Step, Event, Run) has an associated object called a “**Manager**” class (e.g. the “Run Manager” on the previous slide). These modify the particle transport objects and allow them to communicate with one another.
- Each particle transport object also can have an associated “**User Action Class**”. These are the classes that the user creates which are used to extract information from the simulation.
  - Example: We could have a “User Event Action” class which bins energy deposited in a target into a histogram after every event.

Beginning simulation of a CASCADE detector



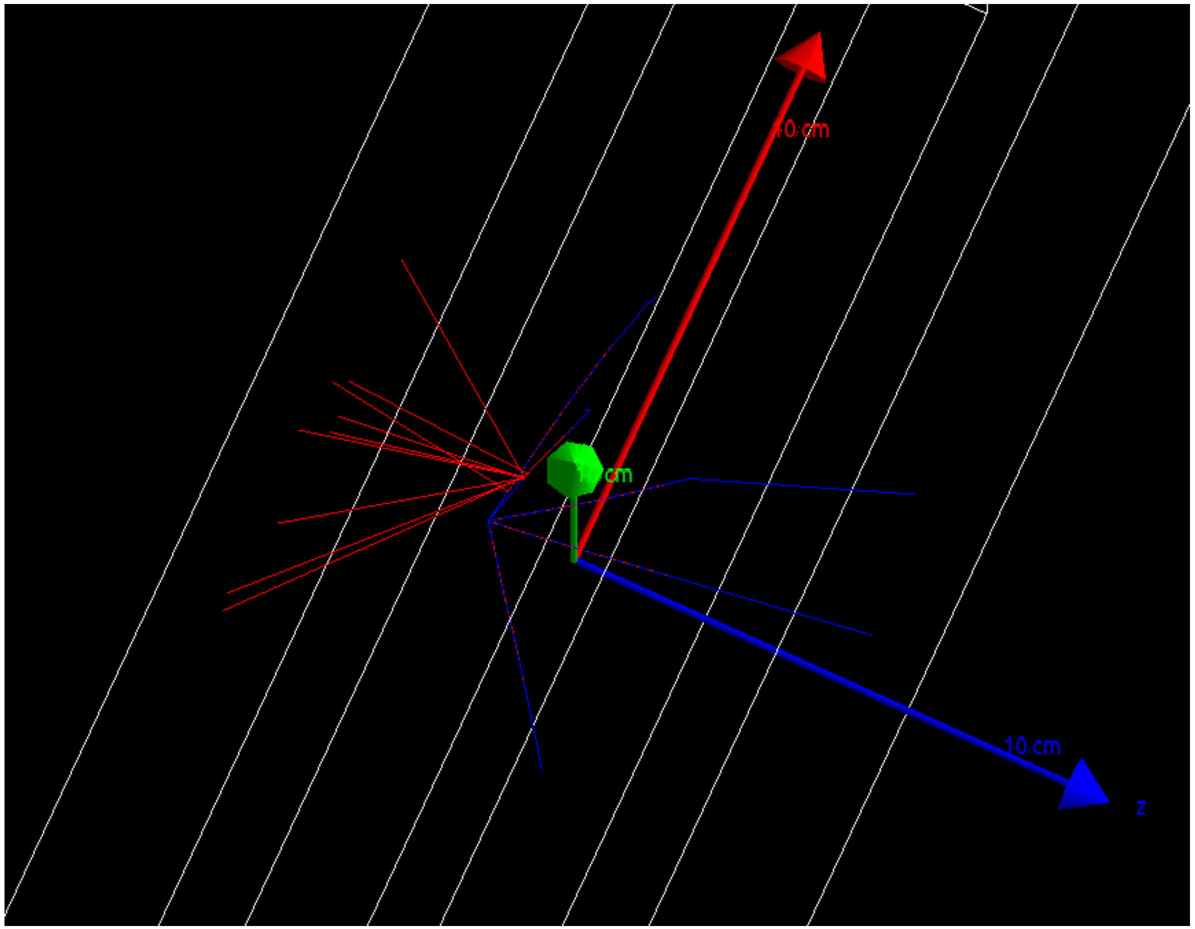
The “Detector” object:

- “Target” is 3x27x27cm box of Ar(85)/CO2(15) gas mixture.
- “World” is box containing this, made of interstellar density hydrogen. (Can’t have zero density materials in Geant4.)

The “Physics List” object:

- For now, using a canned physics list object “FTFP_BERT_EMZ”.
- This is almost certainly not the correct physics list to use. One of the next things I want to do is research the physics list objects.

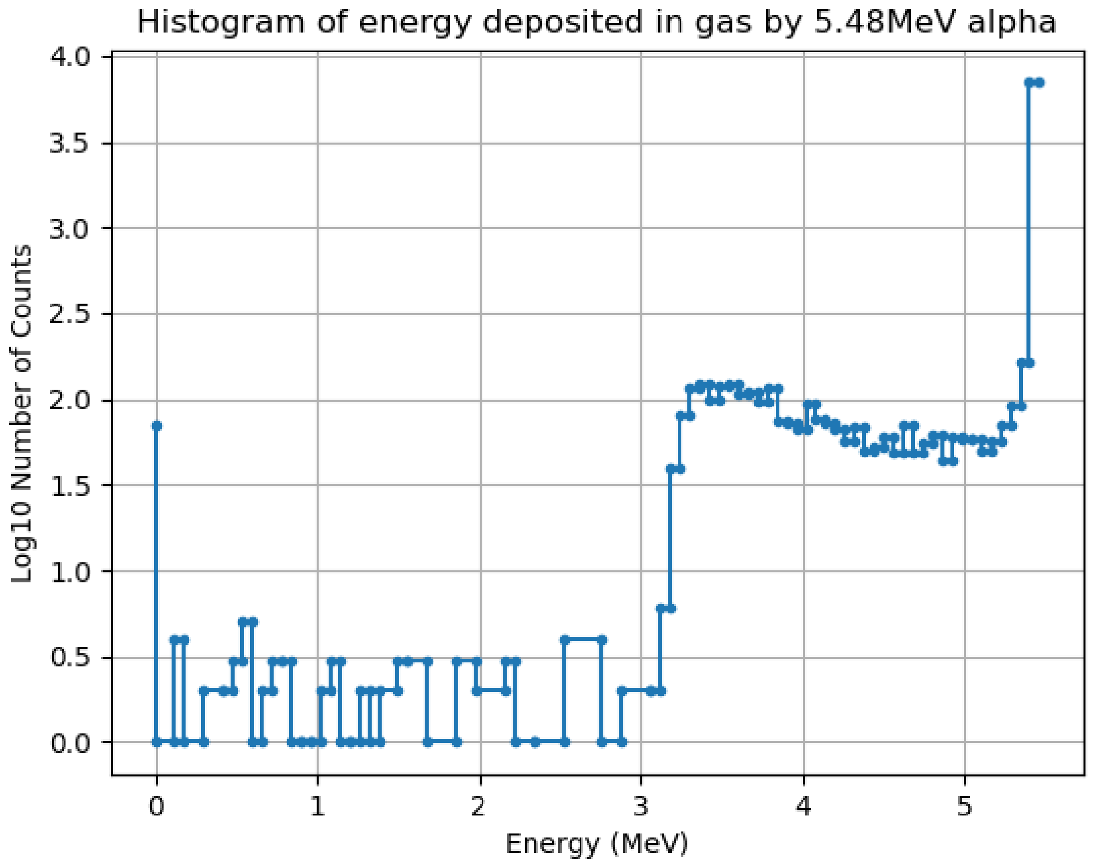
5.48MeV alphas in CASCADE detector



The “Primary Generator” object:

- Alpha particles generated from point source 0.1mm away from target. Energy 5.48MeV, pointing in random direction.
- To the left are five alphas (blue tracks) which ionize the gas and produce many electrons (red tracks). Many means hundreds of electrons per alpha.
- From what I can tell, it seems like the only mechanism by which the alpha loses energy is through electron ionization. This is likely incorrect.

Energy deposited in gas by alpha (albeit with the “wrong” physics)



User Action Class objects:

To get our simulation to output data, we need to include three User Action Classes. The first is a Stepping Action Class which tells checks if the alpha particle is inside the gas or not. If it is, then we start summing how much energy is lost during each step. The Event Action Class then adds this total energy to a histogram at the end of every event. The Run Action Class saves this to a histogram. This is plotted on the left.

Next Steps:

- Find the “correct” physics list:
  - Should include electronic ionization energy loss as well as nuclear scattering energy loss.
  - Should be applicable for alpha particle in this energy regime.
- Add Mylar window in front of gas.
- Calculate the number of secondary electrons produced.
- Add electrode and electric field and propagate electrons to electrode. See how many hit it.
- Add boron-10 foil and simulate energy of fission fragments deposited in gas.

- Probably should not spend too much time on this unless we decide that this is very useful. I would like to get a bit further before with this before I start interfacing with others.
- Most useful thing to do for UCN storage is to understand how physics lists work. We will have to write our own physics lists to include UCN interactions.