MilliQan Simulation

Gabriel Magill
gmagill@perimeterinstitute.ca
(Dated: February 2nd, 2016)

I. INSTALLATION

The first step is to install Geant4, preferably with Qt and the external Datasets. Once that is done you can install the simulation. The link for the Github repository is `https://github.com/talbert48/milliq`. You should clone this repo in a local directory. To install the simulation, you can do:

cd /some_path_outside_the_github_or_geant4_directories/
mkdir BuildMilliQ
cd BuildMilliQ
cmake -DGeant4_DIR=/FullPath/geant4.10.2-build/lib/Geant4-10.2.0/ FullPath/milliq/geant4/
(the first link is where the geant4 build directory is installed, the second is the github folder where the source code of the detector is located)
make
./MilliQ
or you can run it from the macro:
./MilliQ /git_folder/milliq/RootAnalysis/mcp.mac

In order to properly compile/run, You need to access the file milliq/geant4/config/particles.ini. You need to specify the path to the kinematic distribution called hit_4_vecs.txt (it cannot be a .dat file). There is a different distribution for charges 0.001, 0.01, 0.1 and for 48 mass points.

II. GIT FOLDER

A. Root File

In git_folder/milliq/RootAnalysis, there is a root macro called histogram.C which I use to convert the root file to a .dat file to be read in from Mathematica. It shows the names of the ntuples.

B. Macro File

In git_folder/milliq/RootAnalysis/, there is a file called run.mCP.sh and submit.mcp.exclusions.sh. This a big bash script that cycles through all the masses and electric charges, and saves all the output. It is designed for a computer cluster. You can modify it to your architecture.

III. CLASSES

In the following, all the classes are prefixed by MilliQ, and suffixed by .cc. They all have a partner .hh file. The construction of the detector is a bit involved. DetectorBlockLV constructs a block (a scint + pmt). This is called by DetectorStackLV which parametrizes all the blocks into one stack (one layer of the detector) using DetectorBlockParameterisation. DetectorStackLV is called by DetectorConstruction, which parametrizes all the stacks into one detector using DetectorStackParameterisation.

A. DetectorBlockLV

Constructs the scintillator, light guide and PMT from the dimensions provided by DetectorConstruction. Sets the PMT efficiency (detection) spectrum, polished surfaces, reflectivity, R index. Makes PMT and scints sensitive (in the

G4 sense).

B. DetectorConstruction

Calls the StackLV and Stack Parametrization, which in turns creates everything else about the detector. Sets all the dimensions, number of blocks, number of stacks, and the shield. Constructs all the materials. Sets reflectivity, absorption length, fast time constants and all the other detector settings.

C. DetectorStackLV

Creates one stack.

D. DetectorStackParameterisation

Parametrizes all the stacks into the detector.

E. EventAction

Collects all the hits from the scintillators and PMT collections and passes them to Analysis. Obtains Analysis output and fills the .root files event per event.

F. Monopole

Constructs the mCP from sratch. Defines all the quantum numbers.

G. MonopoleEquation

User defined EOM.

H. MonopoleFieldSetup

Used to setup EOM, magnetic field stepper and chord finder for a mCP going through a magnetic field. Currently only works with global magnetic field.

I. MonopolePhysics

Used to specify the charge, mass, and physics of the mCP. As it currently is, you cannot change the mass or electric charge of the mCP without recompiling the program. The charge and masses are scanned using the bash sed command in the Macro File.

J. PMTHit, PMTSD, ScintHit & ScintSD

Gets called upon by Stepping Action when a particle enters the sensitive components. Records the position, time and energy of particles. Packages up everything in hit collections, and passes information to EventAction.

K. PrimaryGeneratorAction

Sets the particle gun, it's momentum, energy and position. Has a function that reads in 4 vectors and coordinates from external files. Can't set the number of events in a run to be larger than the size of the external file ∼25000 events.

L. RunAction

Creates the .root files. There are 3 main containers in the ROOT file. The first replicates the CAEN output. The second is a list of important information. The third is the dE/dx data that is associated to a particle. There is an option in DetectorConstruction (fAlternate = 1) that turns off all the geometry and outputs that dE/dx data.

M. UserEventInformation

Keeps track of number of photons emitted.

N. Waveform & Analysis

These two subsections go hand in hand. The data output in the ROOT file via RunAction is calculated here. We describe in detail what is output. The quantities in bracket shows the length of each vector, and times are in ns, energies are in MeV, and charge is in pC.

We describe what is present in the MilliQCAEN output of the ROOT files. For every channel, we output the following:

- std::vector⟨G4float⟩ TriggerCount(#channels): Set to 0 for each channel
- std::vector⟨G4float⟩ TimeCount(#channels): Set to 0 for each channel
- std::vector⟨G4int⟩ EventID(#channels): Set to 0 for each group
- std::vector⟨G4int⟩ TDC(#channels): Set to 0 for each group
- std::vector⟨G4float⟩ WaveMax(#channels): given the waveform, find the maximal voltage (mV) and output it
- std::vector⟨G4float⟩ WaveMin(#channels): given the waveform, find the minimal voltage (mV) and output it
- std::vector⟨G4float⟩ ChargeIntegral(#channels): Set the charge for each channel

In the above, there are 16 channels, samIndex $\in [0-7]$ (group index) and chanIndex $\in [0,1]$. The mapping between [samIndex][chanIndex] and the channels is like: [0][0]=C1, [0][1] =C2, [1][0]=C3, [2][0]=C5. A waveform as output by CAEN is a collection of ADC counts per time bin. Since the resolution of CAEN is 12 bits, the Vpp is 2.5V, and 1ADC=$2.5V/(2^{12}-1)$=0.61mV, the waveform corresponds to the discretized voltage as a function of time. In our G4 simulation, we have photons hitting the photocathode at given times with a specified quantum efficiency. We convert this to a waveform as follows:

1. Divide the time of one event (one mCP particle) into discrete δt slices.
2. For each δt, count how many photoelectrons are present. Call this n_e.
3. Let G be the gain in the PMT, R the internal resistance, $t_{current}$ the characteristic time it takes for $G \times n_e$ electrons to be measured by a voltmeter. Calculate the voltage as $V(t) = R \times (n_e q_e G/t_{current})$ for each δt.

In the MilliQCAEN output, WaveMax and WaveMin are the maximal and minimal voltages respectively, and the total charge accumulated ChargeIntegral is $Q_{\text{tot}} = \sum_j (V_j - \text{avgBaseline})(\Delta t/R)$, where avgBaseline is the waveform for a channel summed over j=chargeStartCell[channel id] → chargeStartCell[channel id]+16, and divided by 16. By construction, the output charge should equal $Gn_e q_e$. In the limit of $RC >> \tau_{sc}$, we should have the relation that $V_{max} = Q/C$.
We now describe what is output in the MilliQAll output of the ROOT file. For each event, we output the following:

- std::vector⟨G4int⟩ ActivePMT(#activePMTs): A list with ID of all active PMTs
- std::vector⟨G4int⟩ PmtMedianHitTimes(#channels): Gives median hit times of all PMT hits in a channel
- std::vector⟨G4double⟩ PmtAllHitTimes(Total#PMThits): Gives all pmt hit times
- std::vector⟨G4double⟩ TimeOfFlight(#channels): Gives median PMT hit time minus scintillator first hit time in a channel
- std::vector⟨G4double⟩ TotalEnergyDeposit(#channels): Gives total energy deposit in scintillator channel
- std::vector⟨G4double⟩ NumberPMTHits(#channels): Gives number of PMT hits in a channel
- G4int FirstHitScintillator: Gives ID of first scintillator hit by mCP particle
- G4int PhotonCountAllScintillators: Gives total photon count across all scintillators in an event

IV. CONFIGURATION FILES

In milliq/geant4/config, there are a number of configuration files.

A. PMT

Used for PMTs information, specifies: OuterRadius, Radius, CathodeRadius, CathodeHeight, CathodeDepth, Length, HousingReflectivity, PhotonEnergies Efficiency spectrum, ReR and ImR, PMT Resistance, PMT Gain, PMT CurrentTime, RIndex, AbsLength for a variety of PMTs.

B. Geometry

Used for detector information, specifies: Version, NBlocks_X, NBlocks_Y, NBlocks_Z, NStacks, BetweenBlockSpacing_X, BetweenBlockSpacing_Y, BetweenBlockSpacing_Z, Offset_X, Offset_Y, Offset_Z, X Y Z length, LightGuideLength

C. Scintillator

Used for scintillator information, specifies: Density, CarbonContent, HydrogenContent, PhotonEnergies, FastScintOutput, RIndex, AbsLength, ScintillationYield, ResolutionScale, FastTimeConstant, FastScintillationRiseTime, SlowTimeConstant, YieldRatio, BirksConstant.
Used for housing and light guide information, specifies: PhotonEnergies, Efficiencies, Thickness, Reflectivity.

D. Others

particles.ini, specifies: MagneticCharge, ElectricCharge, MonopoleMass, FileName (where particle distributions are located), PathName (where FileName are located), coincidenceThreshold (for online trigger).
mcp.ini, specifies: Number of events in a run.

V. CONCLUSION

This simulation was created by Gabriel Magill (Perimeter Institute for Theoretical Physics) and James London (Ohio State University). If you have any questions, please contact gmagill@perimeterinstitute.ca.