Fermilab

In-Time filtering updates, and ideas on mixing modules in simulation and LArG4 restructuring

Wesley Ketchum

17 January 2017

Working on improvements for in-time cosmic generation

- Current problem: simulation is very inefficient and not robust
  - CORSIKA takes long time to produce events (~7 s in uboone)
  - Filter to select events with particles within very short time window → miss a lot of slightly later light
    - For a trigger, it's the light not the particles that matter…
  - LArG4 takes a while to run over full event
    - Which is why the above filter is tighter than it should be
- We're working out way to try to speed this up, and this involves some code changes
  - Also, going to talk about workflow and use of existing modules, as that's maybe helpful to people

Step 1: Speeding up CORSIKA

- Matt Bass and I discovered bulk of CORSIKA time taken up in reading database entries for showers/particles in showers
- And … bulk of entries in shower databases have no particles
  - They don’t point to TPC and will never lead to detectable particles
  - There to get normalization right, basically
- Matt Bass made DB entries with null showers removed → 50x faster
- Problem now: way of doing it now not random enough
  - Computes n_showers based on flux and time period ***and always pulls same number of showers for each event***
- **Proposed: add Poisson fluctuations on number of showers using RNS**
  - Feedback: should this be an option? On by default?
- ***We are still validating n_particles looks ok***
  - But n_showers in generation looks great

Filtering updates

- FilterGenInTime_module currently rejects/accepts events based on presence of particle in specified time range

- **Proposed: options for sorting particles into separate collections by time**
  - SortParticles switch
    - Defaults false
    - If true, produce two MCTruth collections with instance labels “intime” and “outtime”
  - AlwaysPass switch
    - Default false
    - If true, always pass the event (useful if you only want to sort)

Then, using existing utilities

- More efficient in-time simulation possible:
  - Run G4 *only* over the “intime” collection
    - InputLabels option to select specific MCTruth collection
      - Recall, LArG4 by default uses ALL MCTruth collections
  - Run FilterSimPhotonTime_module to determine if photons hit OpDets in chosen time window
  - IF pass, run G4 *only* over the “outtime” collection
  - Run MergSimSources_module to merge intime G4 and outtimeG4 collections

MicroBooNE flow as example

- Still doing final studies, but as example:
  - Run CORSIKA (~0.15 s per event)
  - Run FilterGenInTime with “intime” as ~10 us before start of trigger period
    - Pass rate ~25%
  - Run G4 over “intime” collection (~10 s per event)
  - Run FilterSimPhotonTime for 1.6 us over trigger period
    - Pass rate ~20%
  - Run G4 over “outtime” collection (~100 s per event)
  - Run MergeSimSources over G4 collections
  - Drop the separate G4 collections from file
- These are all available in LArSoft → you can do this or something similar
  - You need to do your own optimizations, of course

Q’s for other experiments and what to merge

- Opinions/questions on Poisson fluctuations for CORSIKA generation?
  - Currently, this would be a change to way things are done
- Opinions/counterproposals for sorting option in FilterGenInTime?
  - Currently, default → same behavior
- larsim/feature/wketchum_InTimeCosmic to be merged

Thinking ahead

- Rethinking simulation, and how to better optimize
- Significant amount of time spent in G4
- G4 often needs rerunning
  - Any change in lifetime, E-field, space charge/spatial distortions, diffusion, etc.
  - Core issue: these items have more to do with drifting and simulation of detector conditions, not G4 physics/material interactions
  - Potential to speed-up on-demand processing time by pre-processing interactions through G4, and mixing interactions together over some time distribution
    - Use art mixing modules
    - Requires stable geometry and B-field

How I’d envision this working

- Change in output of LArG4
  - Currently SimChannels and SimPhotons are electrons/photons *at the detecting devices*
  - Switch to store energy depositions in the LAr (3D space, time, energy)
    - Associated to parent MCParticle

- “Mixing” module
  - Read in LArG4 outputs from different classes of interaction
    - Use art mixing modules → randomized access from file/SAM dataset
    - Distribute interactions/energy depositions in time as desired
      - Neutrinos and dirt/rock interactions distributed according to beam timing
      - Cosmics random over time window
        - In-time cosmics: force one to be in desired time!
  - Output: complete list of time-distributed particles and energy depositions

How this could work, part 2

- Ionization/Scintillation module
  - Input: energy depositions
  - Output: n_electrons and n_photons produced at that point in space
- Electron drifting module
  - Input: n_electrons at point in space
  - Output: SimChannels or RawDigits
    - Apply space charge effects, diffusion, and drifting time, and E-field at wires
- Photon propagation module
  - Input: n_photons
  - Output: SimPhotons
    - Do photon lookup library or photon propagation simulation

Notes on that

- All of that *except mixing module(s)* is restructuring of existing code
- Should go hand-in-hand with other LArG4 work
- Want ability to include photon propagation as part of G4 jobs
  - → See SimPhoton distribution, and allow you to apply filter on events or find times when filter would pass
- Modules need not produce saved output at each stage
  - Provide options for dropping n_electrons/n_photons or other intermediate outputs as you go through event
- This would imply rethinking BackTracker modules
  - Still want to store Sim::IDEs as part of SimChannel? Or do contributions through associations?
    - Rethink of structure here
    - Continue to keep in mind non-wire SimChannels

End goals of this

- Ability to generate more complete set of reusable G4 interactions in <2 GB memory
  - Less filtering on particles likely to be detectable
  - Make maximum use of computing resources for generating interaction libraries
  - Allow movement in time
- Provide faster turnaround on simulation samples
  - Allow easier modifications to space charge, lifetime, other detector conditions
  - Allow alternative recombination models without needing to rerun G4
  - Avoid lock-in to event composition and fluxes
- Needs to be proven, but seems realizable in short term

Discussion…