Useful GLG4sim tricks

Glenn Horton-Smith
KSU

MAND-sim workshop, June 2005

Ways of controlling GLG4sim: the usual assortment of mechanisms

- Environment variables: only one!
  GLG4DATA = where to find config/data files
- Config/data files: just two main ones
  - materials.dat = material properties (mostly optical)
  - settings.dat = parameters (dimensions and flags)
- User interface commands:
  - LOTS!!!

Where to find documentation and source code

http://neutrino.phys.ksu.edu/~GLG4sim/

Classes that read data files

- GLG4param: reads file like settings.dat – override with `/glg4debug/glg4data`
- GLG4InputDataReader: reads materials.dat – override with `/glg4debug/LoadMaterialsPatchFile`
  [Oops, didn't get ported over from KLG4sim yet!]
- Default detector construction code also reads pmt coordinates from file in $GLG4DATA. Your code might choose to, too.

Classes for scintillation, photon detection, and scattering

- GLG4Scint – makes optical photons, with optional weighting for speedup.
- GLG4PMTOpticalModel – models optical photon detection / absorption / transmission / reflection at a PMTs thin photocathode surface.
- GLG4OpAttenuation – adds Rayleigh scattering to the "attenuation" by any medium, using user-specified lookup table for scattering/absorption ratio. (No calculations.)

Classes for clever event generation tricks: filling and painting, external generators, mixing, chopping.

- GLG4PrimaryGeneratorAction – master mixer
- GLG4DeferTrackProc – splits events that exceed window
- GLG4PosGen_PointPaintFill – fixed point, or random point on a surface or in a volume
- GLG4PosGen_Cosmic – incident on world with random impact parameter
- GLG4VertexGen_Gun – fixed particle type, energy, multiplicity, and random or fixed momentum and polarization
- GLG4VertexGen_HEPEvt – read event info from an external file.

Abbreviated radioactive decay example using Geant4's “RDM”

```
# set "User-controlled: test gun"'s position
  generator to generate events
# on 0.1 mm "skin" outside scintillator volume
/generator/pos/set 9 "0 0 0 paint ! 0.1"

# set "User-controlled: test gun"'s vertex
  generator to generate
# Thorium 232 nuclei at rest
/generator/vtx/set 17 "Th232 0 0 0  0"

# set rate in balloon film to 1 per 1E11 years
  ~ 0.3E-18 Bq
# this is so each nucleus has time to decay
/generator/rates 3 0.3E-18
```

Abbreviated radioactive decay example using external event gen.

```
...

# set Th radioactivity generator
/generator/vtx/set 4
  "generators/Radioactivity/th_generator/th_de
  cays 2000000|"

# set rate in balloon film (can be anything)
/generator/rates 16 0.0057
```

(what are all those integer codes?)

A) Something that needs to die.

B) Indices to fixed tables of position generators, table of vertex generators, or table of position/vertex/rate tuples.

C) See the “information on generators” note.

D) Try the /generators/list and /generators/rates commands.

Class for shortcutting neutron thermalization

- GLG4NeutronDiffusionAndCapture
- This class takes thermal neutrons “away” from Geant4 and adds an exponential time step and Gaussian spatial jump to the neutron, then forces an immediate capture:

```
G4double capture_time= -log(1.0-G4UniformRand())*mean_capture_time;
G4ThreeVector position_offset( G4RandGauss::shoot(),
                               G4RandGauss::shoot(),
                               G4RandGauss::shoot() );
position_offset*= sqrt((slow_diffusion_const * capture_time
                       +fast_diffusion_rms*fast_diffusion_rms)/3.0);
```

Class for shortcutting neutron thermalization, cont.

- Not a good thing to do if your detector is small: you'll get capture on Gd in places where no Gd is! Use /process/inactivate to stop it.

```
/process/inactivate NeutronDiffusionAndCapture neutron
```

- Warning: I've heard that this (Geant4) command may not give you proper warning if you misspell the process name! Watch out!

Classes for special tasks

- GLG4DebugMessenger – mostly harmless diagnostics, plus a few powerful controls.
- GLG4SteppingAction – keeps an “illumination map” and some execution profiling information when compiled with G4DEBUG set.

A motley lot of /glg4debug/ commands

- /glg4debug/dumpelem
- /glg4debug/dumpgeom
- /glg4debug/dump_illumination_map – only if compiled with G4DEBUG=1
- /glg4debug/dumpmat
- /glg4debug/dumpNeutronCrossSections
- /glg4debug/glg4param_dump
- /glg4debug/testsolid
- /glg4debug/SetRunIDCounter
- /glg4debug/setseed
- /glg4debug/glg4param
- /glg4debug/glg4param_read
- /glg4debug/setmaterial
- LoadMaterialsPatchFile and SetMaterialsFile missing – are they desperately needed?

Some more GLG4sim-specific commands

```
/detector/pmtstyle
/detector/select
/event/doParameterizedScintillation
/event/drawTracks
/event/output_file
/event/output_mode
/generator/chain_clip
/generator/event_window
/generator/gun
/generator/list
/generator/pos/set
/generator/rates
/generator/vtx/set
```

```
/glg4scint/dump
/glg4scint/maxTracksPerStep
/glg4scint/meanPhotonsPerSecondary
/glg4scint/off
/glg4scint/on
/glg4scint/reemission
/glg4scint/verbose
/glg4vis/camera/reset
/glg4vis/reset
/glg4vis/upvector
/glg4vis/viewer/reset
/PMTOpticalModel/luxlevel
/PMTOpticalModel/verbose
```

Some things I will try to explain by example

- Scintillation process control
- Geant4 cuts control
- GLG4sim physics list control via GLG4param
- Advanced generator usage
- Extended PMT hit information

Examples

- `/glg4scint/off` when you don't need PMT hits. (Fast simulations.)
- `/glg4scint/meanPhotonsPerSecondary` when you need hits but have too many opticalphotons to track, and `/run/particle/setCut` when default cuts give too many secondaries even with scintillation off. (Muons.)
- Event generator tricks. (External event generators; time correlated events.)

Example 1: turn off scintillation and Cerenkov light

1. Rename co60_z_calibration.mac to co60_z_calibration_noscint.mac
2. Add the following line after /run/initialize and before the first /run/beamOn:
   /glg4scint/off
3. Increase the number of events from 25 to 250, and change the output filename.
4. Run GLG4sim with this macro. Notice the difference in execution time!
5. Open the output file and plot a few of the scintillation summary values from mcevent_tree: fScintillatorEnergy, and fScintillatorCentroid[3].
6. Try viewing some events in the PMT hit display: there should still be some hits, because of Cerenkov light.
7. Now add the following line after /run/initialize and before the first /run/beamOn:
   /process/inactivate Cerenkov
8. Run again. Any change in execution time?
9. Now look at the output file. No more Cerenkov light!

10. Investigate neutron capture detection efficiency as a function of neutron capture position. Edit your X scan macro to generate one 2.2 MeV gamma per event, and scan from outside the balloon to the center in 100 mm steps, with 100 events at each step. Plot scintillator energy deposit vs. capture position. Also plot scintillation centroid position vs. actual position. What effects are and aren't included in this study?

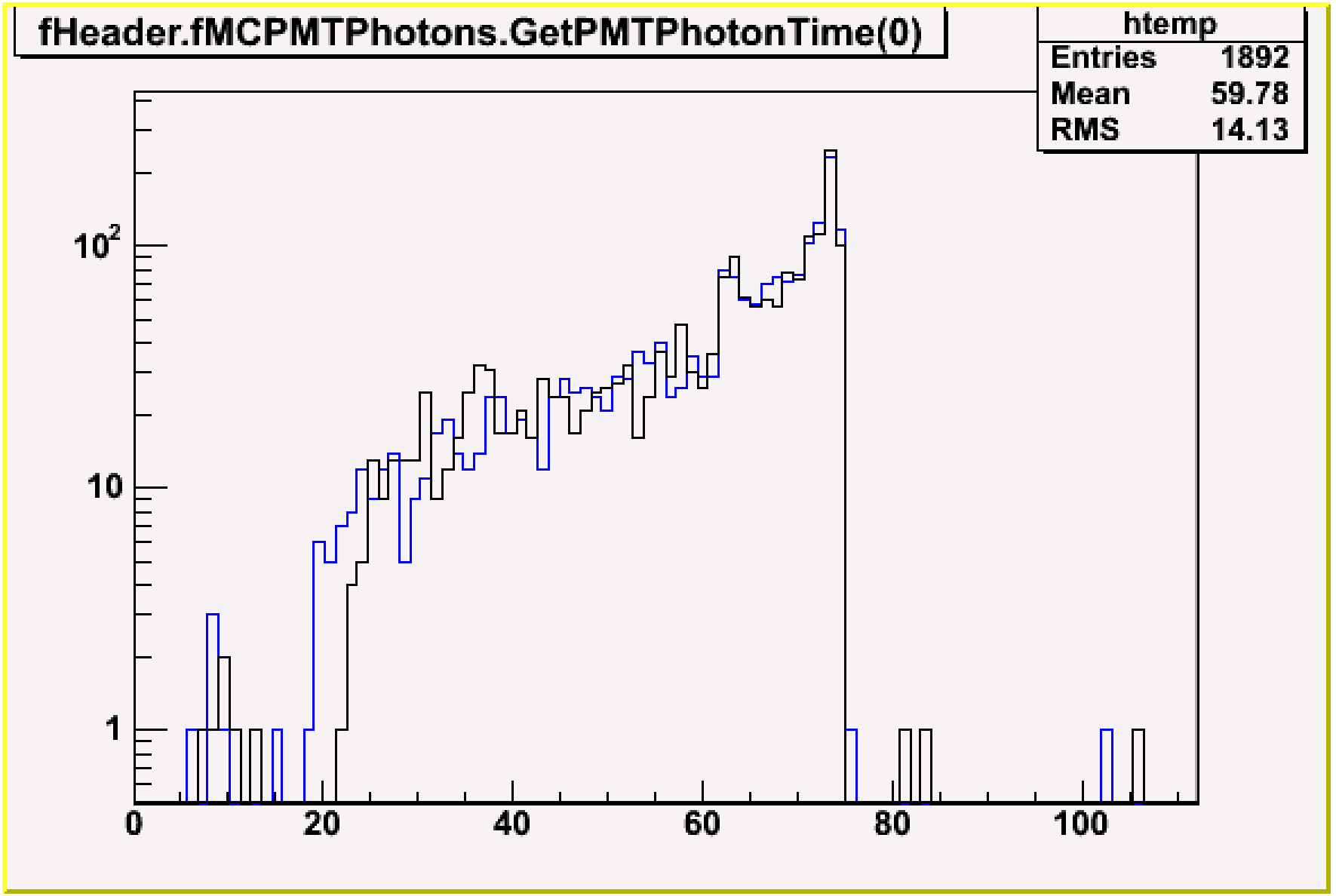
Example 2: reduce number of muon secondaries without reducing muon energy loss

1. Edit muon_test_fast.mac, reducing number of events from 10000 to 1000 so that it will run in a few minutes. (Usually takes ~15 minutes.) Run it.
2. While waiting for your job to finish, look at the macro a little more closely. Note the line "/run/particle/setCut 20.0 mm" before /run/initialize. This limits the secondaries produced by the muon to have energies large enough that their range in the current medium (scintillator) will be over 2 cm (=> ~ 4 MeV for e+/e-); loss due to production of lower energy secondaries will be treated like continuous ("along step") energy loss. This is necessary to speed up muon simulation – the default cut of 0.010 mm was chosen to make sure we get nonlinear scintillation contributions from secondaries right for low energy events.
3. Look at the output from muon_test_fast.mac. Since this was a "no opticalphotons" simulation, it's pretty useless. The only thing you can plot of any interest is deposited energy. Go ahead and do that, just to see. By customizing user stepping action (see custom_hbook.cc for an example), you could collect more interesting data, such as exiting muon angle and position. By adding more processes to the PhysicsList, you could also study neutron and isotope production in various Geant4 hadronic physics models. ** Call for volunteer: someone could figure out how to use "G4PhysicsConstructor"s from Geant4's "physics_list" directory, try out QGSP and LBE hadronic physics lists. **

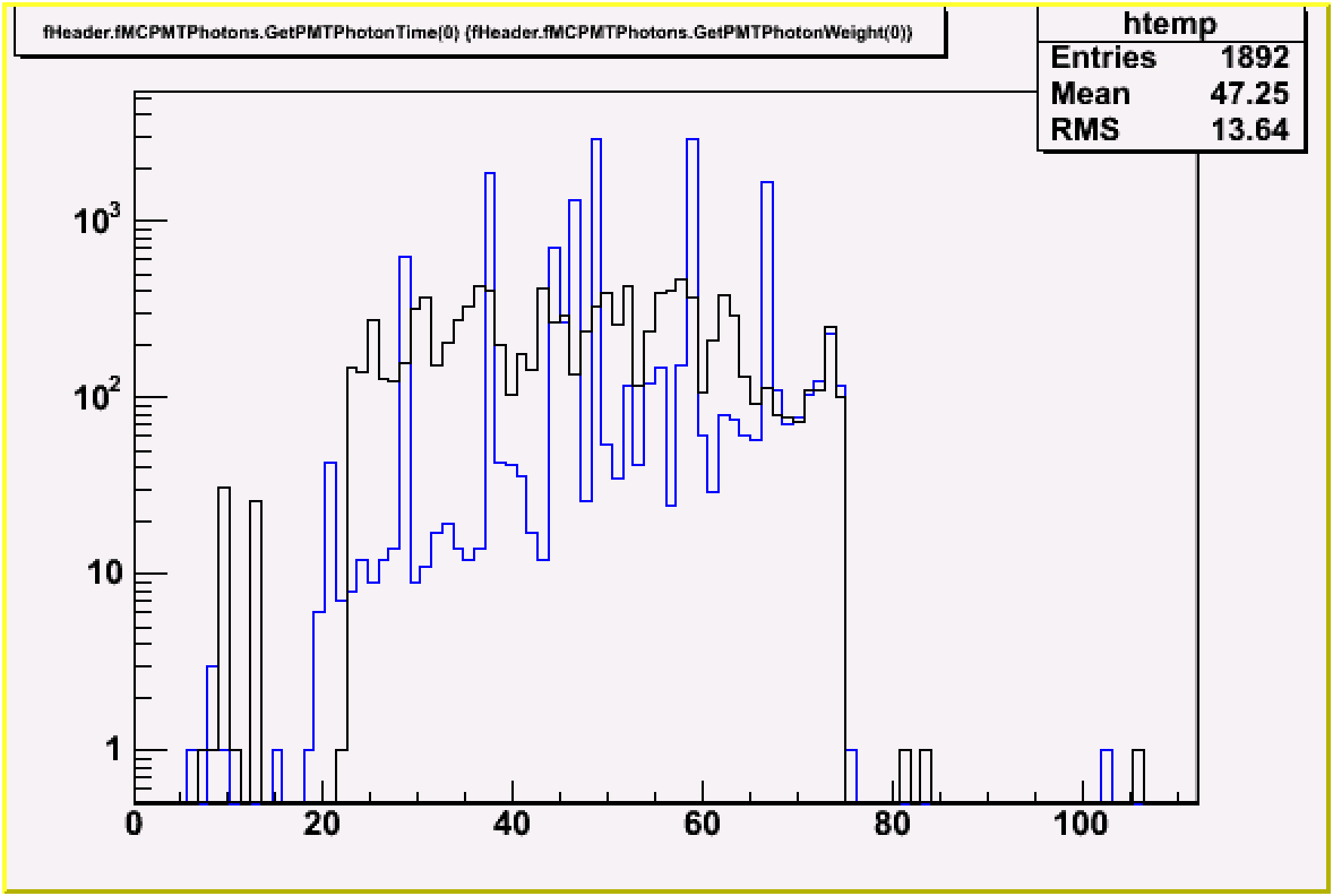
Example 3: reduce number of optical photons tracked without reducing number of optical photons

1. Look at muon_test_slow.mac. This usually takes ~15 minutes to generate just that single track. So don't run it.
2. The important new line here is “/glg4scint/maxTracksPerStep 100”. This tells glg4Scint that no matter what amount of energy is deposited on a given step, it should only generate at most 100 optical photon tracks for that step. It will increase the weight of those tracks as much as it needs to in order to make the mean of the weighted sum come out right. Whether this actually speeds anything up, and how much, depends on whether it would have made more than 100 track on a single step (of a charged particle) in the first place.
3. Alternatively, or in addition, one can use “/glg4scint/meanPhotonsPerSecondary”, which will tell glg4Scint explicitly to start by trying to generate tracks with the given weight. (If this leads to too many tracks, then maxTracksPerStep takes over.)
4. The former method allows low weight tracks to be made on small energy deposit steps; the latter does not, but has the advantage that it's a little more clear what it is you're doing. Neither one speeds up the simulations tracking of Cerenkov optical photons, which actually dominates the execution time here.

Time of first arrival on ID tubes for 1 event (1892 of tubes in ID) using maxTracksPerStep=100 [default is 18000] (blue) and meanPhotonsPerSecondary=100 [default is 3] (black).



Same plot, still time of first hit arriving at tube, but with hit weight applied. (Yech.)



Example 3: reduce number of optical photons tracked without reducing number of optical photons (continued)

5. If we had a good parametric model for the detector time and charge response to a charged particle track, then the right thing to do would be to run the “fast” (no optical model) version to simulate the muon multiple scattering, showering, etc., and use the parametric model to get the hits.

Example 4: generate hits without tracking optical photons

1. Edit your muon_test_fast.mac again, and add the following line before /run/beamOn: “/event/doParameterizedScintillationHits 1”, and run.
2. Look at output – plenty of hits, and hit pattern and total charge look sorta okay, right? Now look at timing: all -1 ns. Anyone interested?

Example 5: delayed coincidence as one event or two?

1. Look at neutron_test.mac: note line
   /generator/event_window 1000000
2. Modify macro to turn on scintillation, and reduce total number of events to 100.
3. Run.
4. Look at time distribution of hits in first entry in mcevent_tree. (First GLG4sim "event".) Now look at second mcevent. Note prompt neutron thermalization signal and delayed capture.
5. Change the macro to set the event_window to 1000, re-run, and compare.

Example 6: delayed coincidence as two events or a lot?

1. In same macro, note commented-out line:
   #/process/inactivate NeutronDiffusionAndCapture
2. Uncomment that line, and run.
3. Look at vertex data for first, second, third, ... events. See what happened?
4. Change the macro to set the event_window back to 1000000, re-run, and compare.
5. Set the event_window back to 1000, and add the word "neutron" as a second argument to the /process/inactivate NeutronDiffusionAndCapture line. Better?