Generate the samples

Thomas Kosc, DUNE-FR workshop
April 2023
kosc.thomas@gmail.com

Simulation workflow

Event generator
- Single Generator
- Supernova (marley)
- Cosmics
- GENIE (beam, atm…)
- …

G4
- Larg4Main (handles interface with G4)
- Energy deposition (ion + scint.)
- Drift of electrons

DETSIM
- Signal simulation at the strip level (rawdigits)
- Noise model

RECO
- Coherent Noise Removal
- Signal filtering, deconvolution
- Hit Finder
- High level reco (showers, tracks…) & event classifier (CVN)

My talk

Thibaut and Laura’s talks

FHiCL files

- **F**ermilab **Hi**erarchical **C**onfiguration **L**anguage. See introduction at (need fermilab account) : https://cdcvs.fnal.gov/redmine/documents/327

- I encourage people to read more at : https://cdcvs.fnal.gov/redmine/projects/art/wiki

- Text file to provide full configuration (i.e modules to call + associated variables) to run art with command `lar -c your_config.fcl -n 5`

  Number of events to generate or treat.

- Few things to know at the start :
  - List of name-value with following syntax → **name: value** (separation \t, \n, ‘ ‘).
  - `#include ‘’dummy.fcl’’` directive makes your fhicl file inherits from all configurations set in dummy.fcl (except if this config file contains `BEGIN_PROLOG` at its beginning).
  - When inheriting, one can modify the value of a name by accessing its full key.
  - Last modification wins.

Dummy illustration

- Here's a dummy configuration that basically has nothing in it. Only the four tables defined will be in every confiuration file one will see when using the dunesw.

```
process_name: DUMMY

services: {}

source: {}


physics: {}

outputs: {}
```

```
/sps/lbno/tkosc/postdoc/tuto/dunefr-workshop23(0)>lar -c dummy.fc
%MSG-i MF_INIT_OK:  Early 12-Apr-2023 10:14:13 CEST JobSetup
Messagelogger initialization complete.
%MSG
Begin processing the 1st record. run: 1 subRun: 0 event: 1 at 12-

TrigReport ---------- Event summary -------------
TrigReport Events total = 1 passed = 1 failed = 0

TimeReport ---------- Time summary [sec] -------
TimeReport CPU = 0.007169 Real = 0.015301

MemReport  ---------- Memory summary [base-10 MB] ------
MemReport  VmPeak = 167.399 VmHWM = 13.398

Art has completed and will exit with status 0.
```

- art merely initializes few things and then exits the job.

Very useful tips (at least to me…)

- Again : I didn't invent anything. All available in this talk.

- Dump the full config file (unfolding all includes + variables called), but don't show the prologs and have a **readable configuration file**.
  - `fhicl-dump protodunevd_standardsingle_driftX.fcl > dump.fcl`
  - equivalent to : `lar -c protodunevd_standardsingle_driftX.fcl –debug-config dump.fcl`

Pretty useful for debugging

- Have the perl scripts **searchfcl.pl** and **fcllookup.fcl** :
  - `perl searchfcl.pl FHIL_FILE_PATH singles_dune.fcl`
  - `perl fcllookup.pl protodunevd_minimal_simulation_services:`

- Finds a fhicl file

- Looks for matching caracters in all fhicl files in $FHICL_FILE_PATH
- I often add ' :' at the end to find only the definition place of a given variable, and not everywhere it is called.

Gun Muon (I)

- Take protodunevd_standardsingle_driftX.fcl

- It contains several tables (or blocks)

```
##include "services_dune.fcl"
##include "services_vdcoldbox.fcl"
#include "singles_dune.fcl"
#include "services_protodunevd.fcl"
```

- Some includes (only of types PROLOGS, i.e variable definitions).

```
process_name: SinglesGen

services:
{
  # Load the service that manages root files for histograms.
  TFileService: { fileName: "single_hist.root" }
  TimeTracker:       {}
  MemoryTracker:         { } # default is one
  RandomNumberGenerator: {} #ART native random number generator
  FileCatalogMetadata:  @local::art_file_catalog_mc
  @table::protodunevd_minimal_simulation_services
}

source:
{
  module_type: EmptyEvent
  timestampPlugin: { plugin_type: "GeneratedEventTimestamp" }
#  maxEvents:   10          # Number of events to create
#  firstRun:    1           # Run number to use for this file
#  firstEvent:  1           # number of first event in the file
}
```

- Table 'services' : define here list of services modules (classes globally visible within an art job, such as Geometry).

- Table 'source' : file input type (empty, art-root).

Gun muon (II)

```
physics:
{
 producers: {
   generator: @local::microboone_singlep
   rns:       { module_type: "RandomNumberSaver" }
 }
 simulate: [ rns, generator ]
 stream1:  [ out1 ]
 trigger_paths: [simulate]
 end_paths:     [stream1]
}

outputs:
{
 out1:
 {
   module_type: RootOutput
   fileName:    "prodsingle_protodune-vd_driftX.root"
   dataTier:    "generated"
   compressionLevel: 1
 }
}
```

- Table '**physics**' to define what one actually want to do.
  - producers
  - analyzers
- Table **outputs** : choose output filename etc.

```
physics.producers.generator.PDG: [ 13 ]  # mu-
physics.producers.generator.X0: [ 150.0 ]
physics.producers.generator.Y0: [ 100.0 ]
physics.producers.generator.Z0: [ 150.0 ]
```

- Substitutes fully qualified keys with their desired updated values.

Gun muon – parameters to tweak

- SingleGen_module.cc is here. It contains numerous variables initialized from reading of the fhicl file. Tweakable from configuration file !

```
std::vector<int> fPDG;                 ///< PDG code of particles to generate
std::vector<double> fP0;               ///< Central momentum (GeV/c) to generate
std::vector<double> fSigmaP;           ///< Variation in momenta (GeV/c)
int fPDist;                            ///< How to distribute momenta (gaus or uniform)
std::vector<double> fX0;               ///< Central x position (cm) in world coordinates
std::vector<double> fY0;               ///< Central y position (cm) in world coordinates
std::vector<double> fZ0;               ///< Central z position (cm) in world coordinates
std::vector<double> fT0;               ///< Central t position (s) in world coordinates
std::vector<double> fSigmaX;           ///< Variation in x position (cm)
std::vector<double> fSigmaY;           ///< Variation in y position (cm)
std::vector<double> fSigmaZ;           ///< Variation in z position (cm)
std::vector<double> fSigmaT;           ///< Variation in t position (s)
int fPosDist;                          ///< How to distribute xyz (gaus, or uniform)
int fTDist;                            ///< How to distribute t  (gaus, or uniform)
```

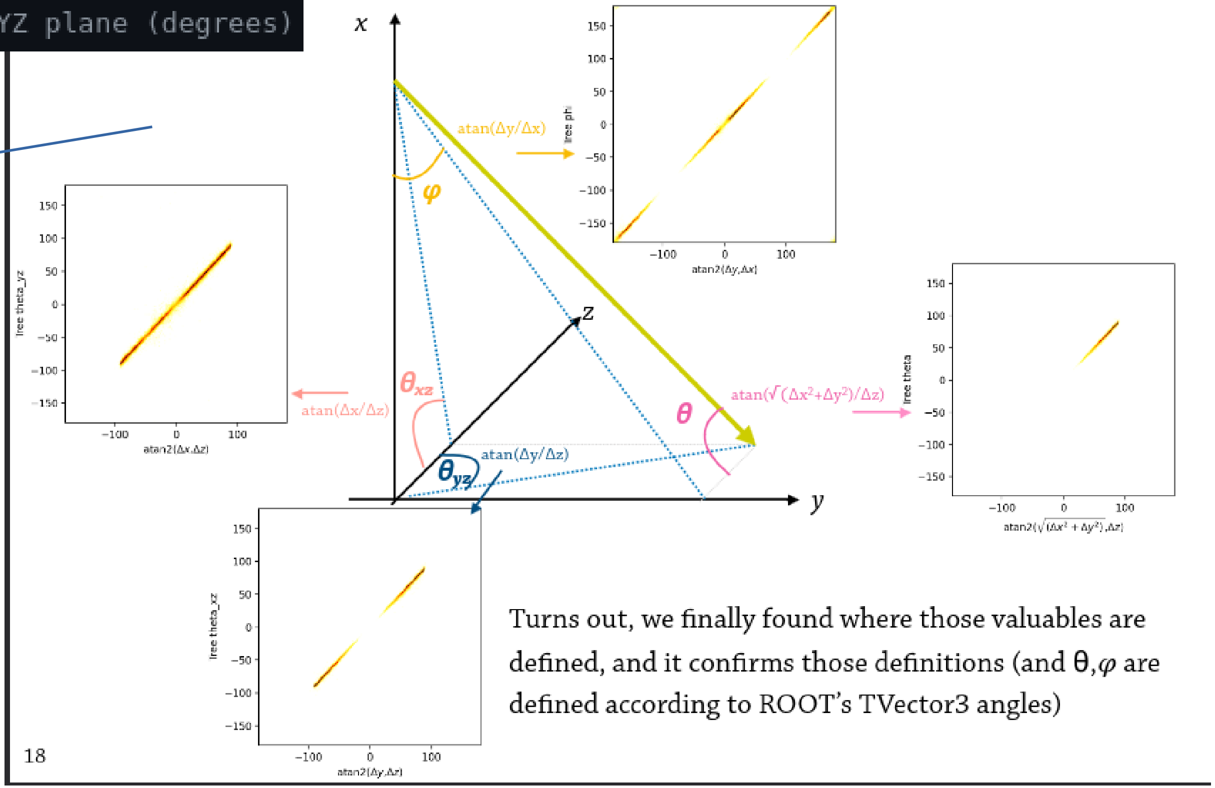
- Use fhicl-dump command to dump the full config file into a local myfcl.fcl, and then tweak parameters wanted (search name ‘generator’).

Angle parametrization

- Direction of generated particle is ruled by two angles : thetaXZ and thetaYZ

```
std::vector<double> fTheta0XZ;      ///< Angle in XZ plane (degrees)
std::vector<double> fTheta0YZ;      ///< Angle in YZ plane (degrees)
```

- From Laura's talk.



- Not the most intuitive ones, a reproducible method is to go for Dx, Dy and Dz and find the corresponance

Cosmics (I)

- Take gen_protodunevd_cosmics.fcl

- Services, source and outputs blocks are identical, except for physics.

```
physics:
{
 producers:
 {
   # cosmicgenerator: @local::protodune_corsika_cmc
   cosmicgenerator: @local::standard_CORSIKAGendp_CMC
   ar39:  @local::protodunesp_39ar
   ar42:  @local::protodunesp_42ar
   kr85:  @local::protodunesp_85kr
   rn222: @local::protodunesp_222rn
 }
simulate: [ cosmicgenerator, ar39, ar42, kr85, rn222 ]
 #define the output stream, there could be more than one if using filters
 stream1:  [ out1 ]

 #trigger_paths is a keyword and contains the paths that modify the art::event,
 #ie filters and producers
 trigger_paths: [simulate]

 #end_paths is a keyword and contains the paths that do not modify the art::Event,
 #ie analyzers and output streams.  these all run simultaneously
 end_paths:     [stream1]
}
```

- Only the physics block is different, and calls and protoDUNE-DP module cosmic generator (relies on corsika).

Cosmics (II)

- Parameters to tweak :
  - ProjectToHeight : where to start the shower [cm]
  - ShowerAreaExtension :
  - BuffBox : extension of acceptance box (default = cryostat) [xlow ; xsup ; ylow ; ysup ; zlow ; zsup] to capture more cosmics.
  - RandomXZShift : shift of the beginning of the shower.

```
standard_CORSIKAGendp_CMC:
{
 module_type:          "CORSIKAGendp"
 SampleTime:            8.0e-3             #integration time in seconds
 TimeOffset:           -4.0e-3             #time in seconds before a spill to begin the int
 ProjectToHeight:       856                #height to which particles are projected [cm]
 ShowerInputFiles:      [
                           "/cvmfs/dune.osgstorage.org/pnfs/fnal.gov/usr/dune/persistent/sta
                           "/cvmfs/dune.osgstorage.org/pnfs/fnal.gov/usr/dune/persistent/sta
                           "/cvmfs/dune.osgstorage.org/pnfs/fnal.gov/usr/dune/persistent/sta
                           "/cvmfs/dune.osgstorage.org/pnfs/fnal.gov/usr/dune/persistent/sta
                           "/cvmfs/dune.osgstorage.org/pnfs/fnal.gov/usr/dune/persistent/sta
                         ] #list of sqlite dbs with corsika showers
 ShowerFluxConstants:   [ 1.72e4, 9.2e3, 6.2e3, 9.2e3, 6.2e3] #list of flux constants per sh
 BufferBox:             [ -300.0, 300.0, -300.0, 300.0, -300.0, 300.0 ] #list of buffer box
 ShowerAreaExtension:   2000               #amount to extend the shower area beyond the cry
 RandomXZShift:         1000               #amount to randomly shift shower start point in
 DoRotation:            true               #perform flux rotation for DP with drift in X
 UseIFDH:               false               #true for jobs at FNAL, false for jobs at CERN
}
```

- Playing with buffer box to increase generated particle acceptance.
- RandomXZShift : typical confusing hardcoded variable HD orientated.
- Name of the module shown here, so source code is : **CORSIKAGendp_module.cc**

What do I have in my art output file (I) ?

- art-root output files are hardly readable with a simple browser in a root sessions. I suggest two options :

- lar -c eventdump.fcl your_art_root_file.fcl
  That command will display on screen the objects and their types that you have.
  - Limited amount of information
  - Relevant for having a large view on what’s going on.

```
PRINCIPAL TYPE: Event
PROCESS NAME | MODULE LABEL.. | PRODUCT INSTANCE NAME | DATA PRODUCT TYPE............ | SIZE
SinglesGen.. | generator..... | ..................... | std::vector<simb::MCTruth>... | ...1
SinglesGen.. | rns........... | ..................... | std::vector<art::RNGsnapshot> | ...1
SinglesGen.. | TriggerResults | ..................... | art::TriggerResults.......... | ...1
```

- This chart gives you many useful infomartion : name of the process, labels of the art objects (you need the to be able to retrieve them, see next slide), and the size.

- Later, one will have a look at high level stuff such as recob::Track. If the size of such an object is 0, you already know you have 0 reconstructed track in this event.

What do I have in my art output file (II) ?

- To go further : one can have a script (use checkProd.cc) that one can launch in a root session. This script will display more detailed information on the simb::MCTruth object.

```
root [1] checkProd("prodsingle_protodune-vd_driftX.root")
Successfully opened file prodsingle_protodune-vd_driftX.root
size = 1
Has 1 particles

Particle pdg 13 and mass = 0.105658
(E;p) = ( 5.00112 ; 0 ; 0 ; 5 ) GeV
has mother ? --> -1
has daughters ? --> 0
```

- art objects are retrieved with this command taking in argument the tag of the objects.

  Convention is '**label :name'.**

```
// get art/larsoft product desired
auto const& mGen          = *ev.getValidHandle< std::vector<simb::MCTruth> >(taggen);
```

Keep it as a skeleton !

- Good to keep such a skeleton script in one's home directory that one can then modify to look at stuff related to his/her current work.

- I added most of the #include one would encounter.

```
// energy deposits
#include "lardataobj/Simulation/SimEnergyDeposit.h"

// raw data
#include "lardataobj/RawData/RawDigit.h"

// Reco Base
#include "lardataobj/RecoBase/Wire.h"
#include "lardataobj/RecoBase/Cluster.h"
#include "lardataobj/RecoBase/Hit.h"
#include "lardataobj/RecoBase/Shower.h"
#include "lardataobj/RecoBase/SpacePoint.h"
#include "lardataobj/RecoBase/Track.h"
#include "lardataobj/RecoBase/Vertex.h"
```

Going to G4 !

- Next stage of the simulation workflow : simulating the particle propagation, ionization + light in the detector.

- Thibaut now leads.

Fixing the seed

- Pretty useful to debug situations. The seed is fixed via the NuRandomService
- Match the names in the service to the ones in physics.simulate

```
NuRandomService: {
   IonAndScint: 1234
   PDFastSim: 1234
   endOfJobSummary: true
   largeant: 1234
   policy: "preDefinedSeed"
   service_type: "NuRandomService"
}
```

```
physics: {
   end_paths: [
      "stream1"
   ]
   producers: {
      largeant: {
         enableVisualization: false
         macroPath: ".:./macros"
         module_label: "largeant"
         module_type: "larg4Main"
         visMacro: "vis.mac"
      }
   }
   simulate: [
      "largeant"
   ]
```

- In this example (gun muon + g4 stage1), I have a reproducible energy deposition from one run to the other

```
std::vector<sim::SimEnergyDeposit>.................................. | 86636
```

Brown cake recipe (French)

- 250 g farine
- 125 g sucre roux (ou moins)
- 120 g d’eau
- 130 g de miel
- 1 œuf
- 1 cuiller à café de bicarbonate de soude
- 2 cuiller à café d’anis (ou plus).

Mélanger œuf + sucre, battre. Ajouter farine (ça fait un gros truc tout moche). Mélanger eau + miel, faire tiédir, et ajouter au mélange. Battre.
Ajouter le bicarbonate de soude et l’anis, mélanger.

Cuisson : 150° (th. 5) ~1h30 pour un plat à cake (graisser légèrement au préalable, ou utiliser papier cuisson).