Producing overlay samples

Adi Ashkenazi (MIT), Wesley Ketchum (FNAL)

Production general concerns

- Alex/discussion hit on many of these really well this morning!

- Can very generally separate generally into two concerns
  - Ensuring your workflows are agnostics to data/MC/overlay as much as possible
  - Data access/retrieval models, and ensuring your workflows are robust enough to account for them

MicroBooNE approach

- Based on its needs, MicroBooNE decided to adopt a general philosophy of starting with a (off-beam unbiased) data event, and running the entire simulation/mixing of interactions *on top of it*

- Every event inherits from the raw data of the original event, and we mix as part of that

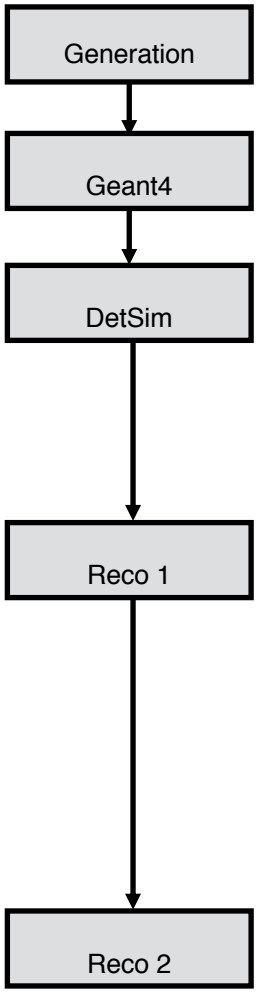
Pros and Cons

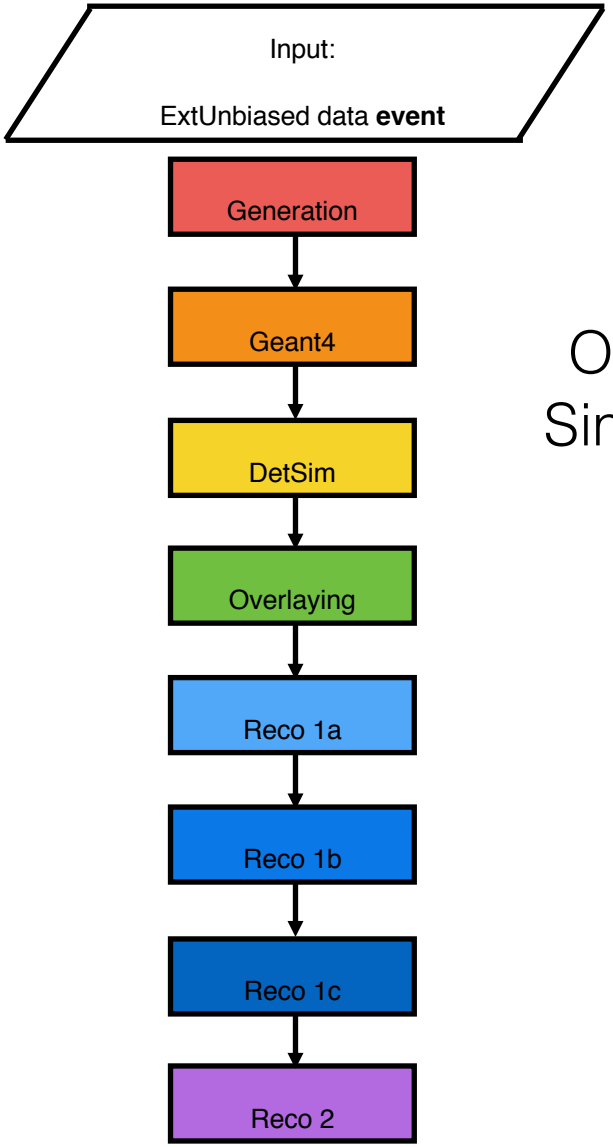
- Pros
  - Every event naturally has a valid run number and event number and time stamp
  - We do not need two datasets: just input data, and run simulation inline following the normal chain
    - Pure neutrino simulation is not that intensive or inefficient…
    - Naturally get subrun/run-level information, like POT
    - ***We run the full chain on a single worker node***
  - Allows for systematic controls on data reuse

- Cons
  - Small data files → small simulation files
  - No reuse of neutrino interactions
  - Need to incorporate time-dependent simulation in-line
    - Calibration is a pre-requisite for launching production!
  - Need special handling in cases to tell the event it has simulation in it
  - Out-of-the-box, filter modules in simulation/generation will lead to big data I/O inefficiencies
    - Will discuss our hack for this …

Overlay production

Typical Sim Chain





Overlay Sim Chain

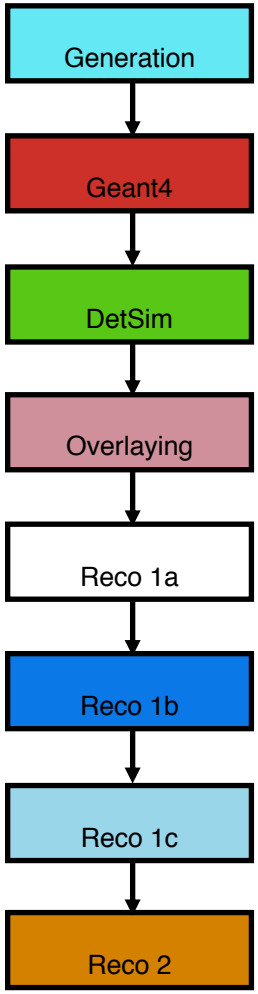
Workflows: Module Labels

- Alex touched on this this morning (and it rang so so true): need to carefully align your module labels!

- Generally easy to do this for simulation steps
  - We don't typically use process name in module labels, so just have to link of the module labels generally → easy to do for 'generator' things
  - We have used a 'mixer' label so we can look at raw collections more easily → all downstream reco needs to look to that, which took a lot of debugging…

- Also, specified a new file type in our metadata: 'overlay'

Overlay sample



uboonecode/fcl/gen/genie/standard_overlay_gen_driver.fcl

uboonecode/fcl/g4/wirecell_g4_uboone.fcl

uboonecode/fcl/detsim/wirecell_detsim_overlay_uboone.fcl

uboonecode/fcl/detsim/standard_overlay_uboone.fcl

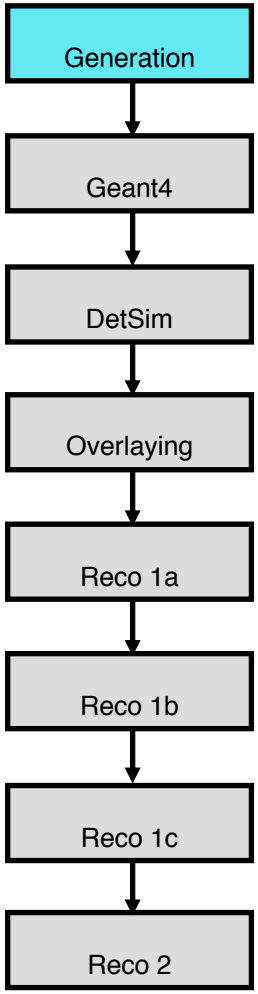
uboonecode/fcl/reco/MCC9/reco_uboone_mcc9_8_driver_overlay_stage1a.fcl

uboonecode/fcl/reco/MCC9/reco_uboone_mcc9_8_driver_overlay_stage1b.fcl

uboonecode/fcl/reco/MCC9/reco_uboone_mcc9_8_driver_overlay_stage1c.fcl

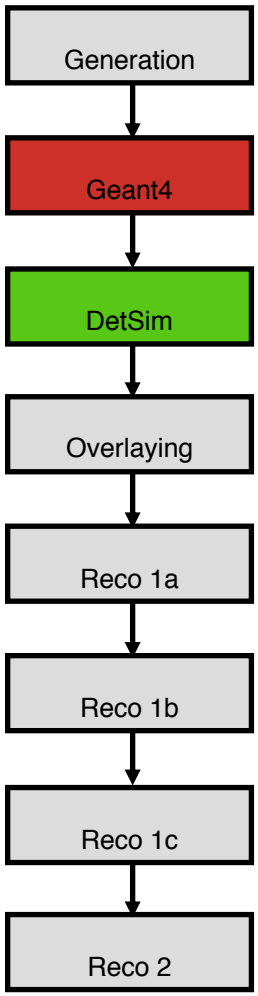
uboonecode/fcl/reco/MCC9/reco_uboone_mcc9_8_driver_overlay_stage2.fcl

Generation



- As mentioned, we start with an off-beam unbiased (not light-triggered) data event
  - In uboone parlance: “EXT unbiased”
- Use standard GENIE generation fcl files, with the exception that we have a RootInput source instead of EmptyEvent
  - Except where we don’t do this … more on that later…

G4 and Detsim

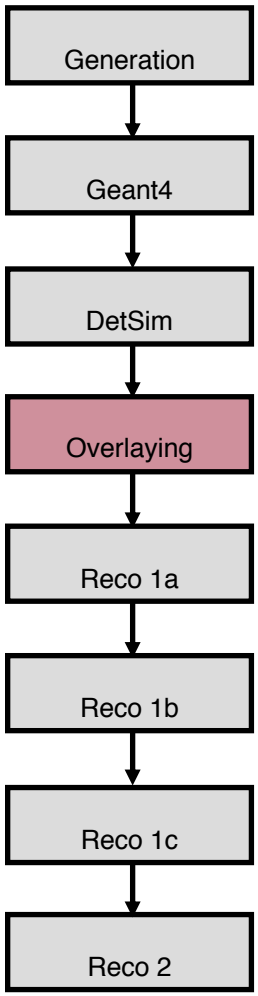


- No G4 differences
- Detsim we are careful to…
  - Turn off noise simulation
  - Find bad/non-existent channels in data (event-by-event) and zero our MC
- Calibration!
  - In the end, we want to apply a purely data calibration to these events
  - Scale MC signal to match that of data

$$\frac{K_{\mathrm{MC}}(x, y, z, \mathrm{plane})}{K_{\mathrm{data}}(x, y, z, t, \mathrm{plane})}$$

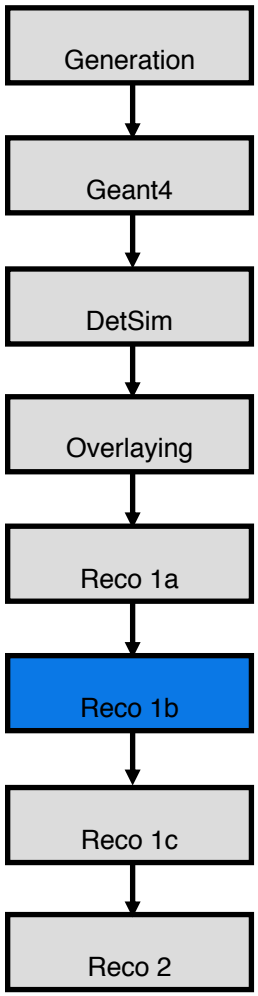
  - Trusts that the convolution of charge is linear…

‘Mixing’



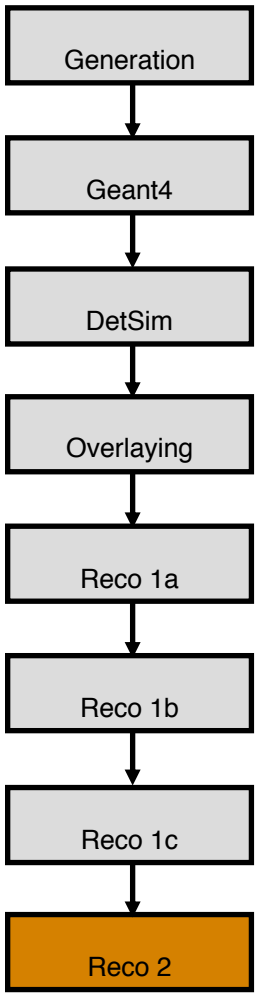
- Here we do the waveform addition
  - Baseline subtract MC and ‘add’ that to the data waveforms
  - Account for saturation/other simulation effects as desired
- For MicroBooNE
  - TPC waveforms are non-zero-suppressed (easy!)
  - PMT waveforms are non-zero-suppressed around neutrino beamgate (easy enough!)
  - CRT hits we just add to it
    - Still early in CRT reco for us, so not too complicated

Signal processing and hit-finding (Reco1)



- Reconstruction is the same for data/MC/overlay
- EXCEPT…
  - I mentioned before about the difference in optical timing → easily handled by algorithm configuration, but this is annoying
- Also mentioned before: the backtracking issue
  - Forces us to do a dedicated 'reco1b' step between reco (1a) and reco1 filtering (1c)
    - Services configured once per job…

Higher-level reconstruction (Reco2)



- Again, mostly identical to data and MC
- Here is where we perform calibration (e.g. dQdx and dEdx) → calibrate like it's data!
  - Again, following along what we've done in the detector simulation stage

But, it's always the special cases…

- We have some generation that runs a filter after GENIE (or even G4)
  - Final state/topology-dependent things, like NCpi0, CCpi0, NC single photon, etc.
  - These are generally rare-ish, which is why we have filters for them

- Problem: if you throw a filter into that chain, art will throw away the whole event
  - How would it know otherwise?
  - → this would lead to unacceptable data I/O issues

And so, we have a special workflow for these!

- Start by generating a bunch of events through the filtering stage (at gen or g4 level) on the worker node in standalone process
  - How many events? As many as the number of data events in our file
  - Which is …
    - (a) we don't know without getting the file first
    - (b) if we had the file, there's not a super-quick way to get the number of events in it I think…
    - (c) even if we had that, there's not a way to tell art to generate up to *n* events passing a final trigger path
  - So … we guess!
    - Data files typically have <50 events, so just make sure to simulate enough for that
      - (not really a problem to simulate too many events → good way to drive up CPU efficiency!)

More special workflow!

- OK, now we have an art-ROOT file with our generated info!
- Need to add into the data event
  - Could use art::EventMixing utilities … except there's not a way to access existing associations
    - Even with that, have to remap the art::Ptrs…
  - OK, so, be hacky and use gallery to read this secondary file, and copy in everything (and remap associations)
    - **SimInfoMixer** module in ubevt/DataOverlay area…
    - Except … gallery as of two months ago couldn't access subrun objects → our POT normalization!!!
    - Even if we had our subrun POT, what if we have too many simulated events corresponding to greater POT and we don't merge all of them → need to prescale POT but again then we need to know the number of events in each file which ***AAHHH!***

Even more special workflow

- Typically we have a POT per subrun …
- So …
  - Have a module to assign a POT per event
  - Now, in the SimInfoMixer module (which takes as input a data event, and reads the simulation event in gallery) read in the POT information and accumulate it event-by-event
  - At the end of each subrun, fill out the accumulated POT object per subrun

- And, finally, we are happy and can proceed with simulation in the overlay chain

Oh wait!

- Also, we hack a way to include the originating generation fcl file into the SimInfoMixer fcl file to retain provenance
  - Remember, we read in the simulation via gallery → there is no provenance information kept!!

- Initial worker script looks like
  https://cdcvs.fnal.gov/redmine/projects/uboonecode/repository/revisions/v08_00_00_br/entry/tools/overlay_scripts/init_gen_common.sh

```
#!/bin/sh

MY_FCL_FILE=$1 #prodgenie_bnb_nu_filtered_NCPiZero_uboone.fcl
MY_N_EVENTS=$2 #2500
MY_OUTPUT_FILE=${3:-genfile.root.local}
MY_OUT_LOG=larInitGen.out
MY_ERR_LOG=larInitGen.err

echo "#include \"$MY_FCL_FILE\"" > local_gen.fcl
echo "physics.producers.generator.FluxCopyMethod: \"IFDH\"" >> local_gen.fcl
echo "physics.producers.generator.MaxFluxFileMB: 500" >> local_gen.fcl
echo "services.IFDH: {}" >> local_gen.fcl

echo "#include \"$MY_FCL_FILE\"" > local_gen_include.fcl
echo "physics.producers.generator.FluxCopyMethod: \"IFDH\"" >> local_gen_include.fcl
echo "physics.producers.generator.MaxFluxFileMB: 500" >> local_gen_include.fcl
echo "services.IFDH: {}" >> local_gen_include.fcl
echo "gen_detail: { physics: {@table::physics} services: {@table::services} outputs: {@table::outputs} source: {@table::source} process_name:
@local::process_name }" >> local_gen_include.fcl

if [ -f $MY_OUTPUT_FILE ]; then
    echo "File $MY_OUTPUT_FILE exists, so not generating again."
else
    echo "File $MY_OUTOUT_FILE does not exist, so running generation of it..."
    echo "Running command 'lar -c local_gen.fcl -T ./genfile_hist.root -o $MY_OUTPUT_FILE -n $MY_N_EVENTS > $MY_OUT_LOG 2> $MY_ERR_LOG' "
    lar -c local_gen.fcl -T ./genfile_hist.root -o genfile.root.temp -n $MY_N_EVENTS
    lar -c standard_overlay_gen_SubRunPOTInEvent.fcl -s genfile.root.temp -T ./genfile_pot_hist.root -o $MY_OUTPUT_FILE -n -1
    rm genfile.root.temp
fi
```

At this point you may be thinking…

…and you'd be right

- This was developed in order to hack something together to get us working
- We think it works for our immediate production needs, but it is very hacky and isn't going to cover everything well
- We should probably invest in a better model (and which could inherit from a lot of existing tools)
  - EventMixing filter provides real access to the underlying art::Event from secondary files it reads?
  - 'Generate *x* events' passing a filter would be nice…
  - < your ideas here... >

Finally: considering the available events

- We use off-beam unbiased data ("EXT unbiased") as a base for overlays

- At this point (naïve → paranoid) we are worried about using the same event twice if there's pathological events
  - In some cases this may be ok: e.g. if those events are being used for BDT training on particular signals, we could probably reuse those for different signals…
  - Also, e.g., could use same event once for BNB and once for NuMI

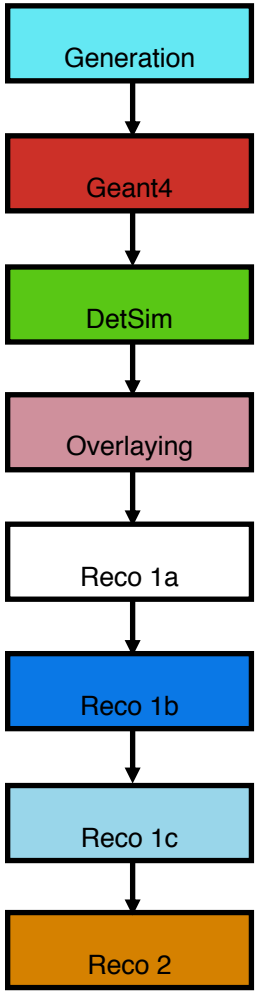
- Note to experiments: think about how much data you need!

Our overlay event accounting…

Run Period	POTs in data	equivalent events	available EXT Unbiased events	Factor in statistics EXT Unbiased / data
I	1.81E+20	1.64E+05	2.5E+06	15.2
II	3.05E+20	2.76E+05	4.8E+06	17.3
III	2.67E+20	2.42E+05	3.2E+06	13.2
IV	1.33E+20	1.20E+05	1.5E+06	12.4

- Note that we probably have enough … but not by a lot necessarily!
- Experiments really need to consider this as part of data-taking plans

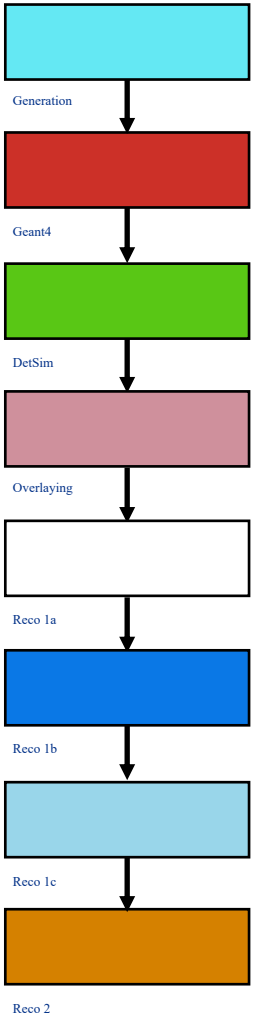
Backup


Generation
Geant4
DetSim
Overlaying
Reco 1a
Reco 1b
Reco 1c
Reco 2

- SS

The Overlay sample

Production workflow



uboonecode/fcl/gen/genie/standard_overlay_gen_driver.fcl

uboonecode/fcl/g4/wirecell_g4_uboone.fcl

uboonecode/fcl/detsim/wirecell_detsim_overlay_uboone.fcl

uboonecode/fcl/detsim/standard_overlay_uboone.fcl

uboonecode/fcl/reco/MCC9/reco_uboone_mcc9_8_driver_overlay_stage1a.fcl

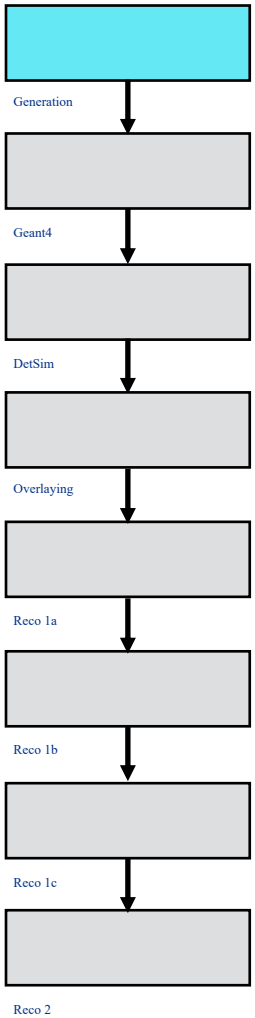
uboonecode/fcl/reco/MCC9/reco_uboone_mcc9_8_driver_overlay_stage1b.fcl

uboonecode/fcl/reco/MCC9/reco_uboone_mcc9_8_driver_overlay_stage1c.fcl

uboonecode/fcl/reco/MCC9/reco_uboone_mcc9_8_driver_overlay_stage2.fcl

Overlay Production

Generation



The Generation stage takes as an input an EXT Unbiased data event

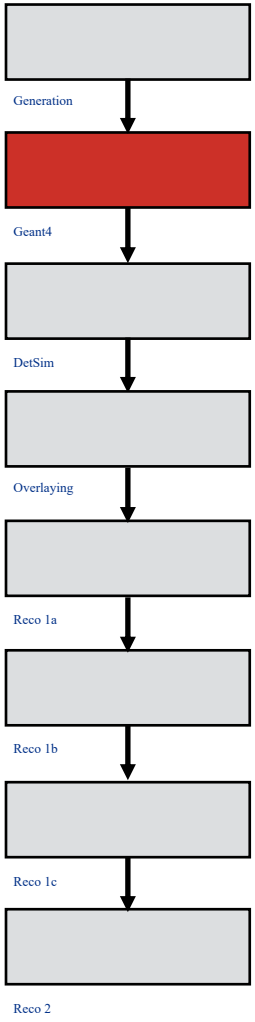
Each simulated event is matched to a data event.

There are two ways to generate the overlay event:

1. generate each event on top of a data event
2. generate many events, filter them, and then match to a data file and matching each event

Overlay Production

Geant4

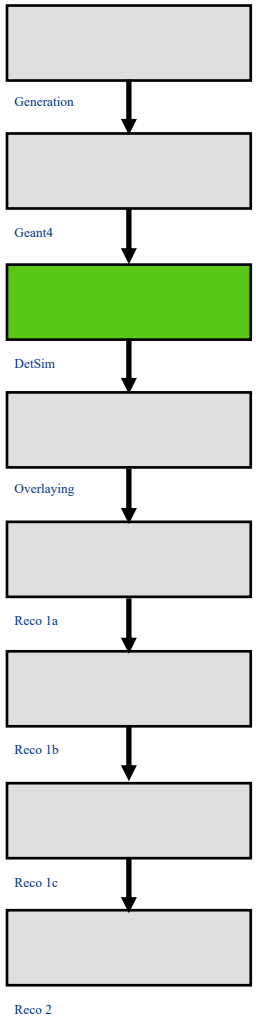


Same as for MCC + keeping the energy depositions

Overlay Production

Detector Simulation no noise

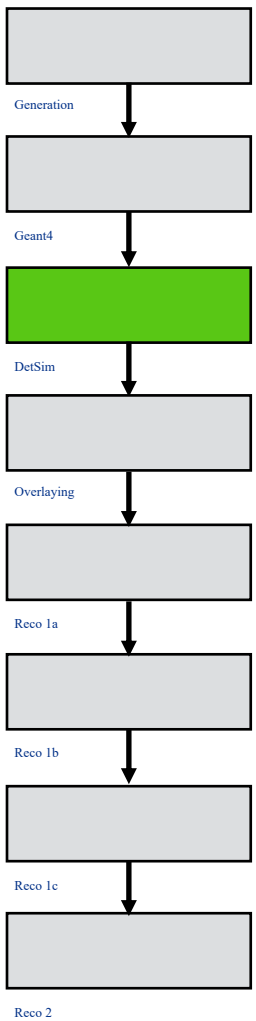
Detector simulation



- Noise filter to find bad channels (without filtering)
- Regular no noise det sim
- Zeroing out the signal in the bad channels
- CRT simulation
- Masking out CRT hits in non available CRT panels for in the input data file
- Calibration (next slide)

Overlay Production

Detector Simulation no noise



Premise: We want to consider the entire sample as data.

Method:

Apply a gain to each generated waveform :

$$\frac{K_{\mathrm{MC}}(x, y, z, \mathrm{plane})}{K_{\mathrm{data}}(x, y, z, t, \mathrm{plane})}$$

These are taken from the calibration maps derived by Tingjun Yang and Varuna Meddage.

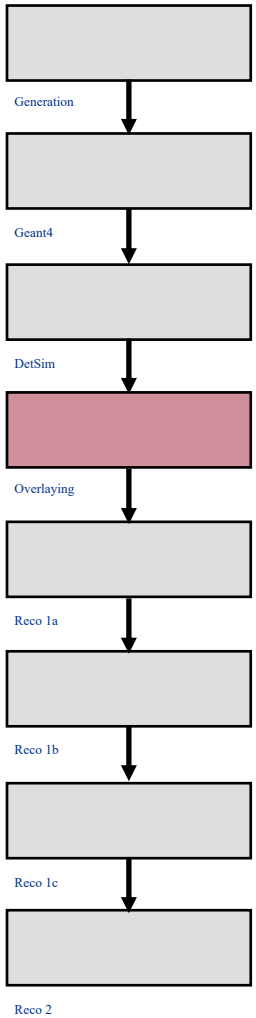
Given that:

- The same energy deposition causes waveforms in all planes
- The convolution of the charge is linear

The gain was applied to each charge on sim wire

Overlay Production

Mixing



This stage is where the overlay is actually being done.

Every generated waveform from a PMT or a TPC wire is simply added to the corresponding one from data.

The generated pedestal is subtracted to be left with the data pedestal

As for the CRT, the reconstructed simulated CRT hits are added to those from data

Overlay Production

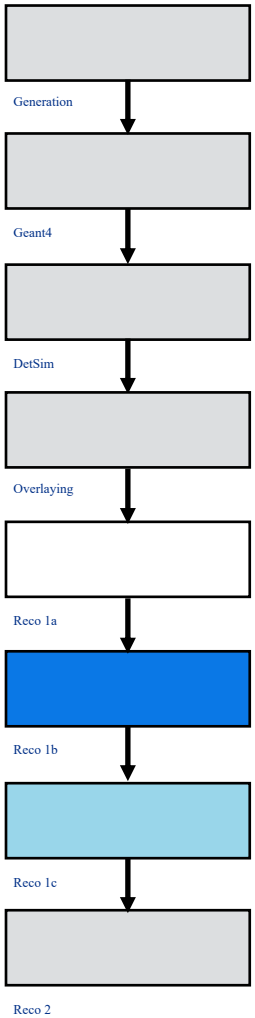
Reco1

Trigger timing:

– As for every hit on a TPC wire the backtracking procedure is looking for a hit in the correlated SimChannel, this procedure requires the timing to be defined using the MC trigger.

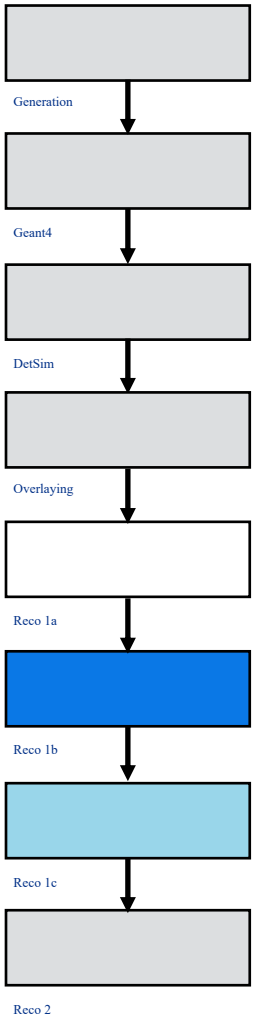
– As all the PMTs waveforms from cosmic data and BNB MC are added, and the time of the added waveform is defined using the external trigger coming from the cosmic data stream

The proper flash timing reconstruction required the timing to be defined using the data trigger.



Overlay Production

Reco1



Therefore, the stage 1 of the reconstruction was sub divided into three different steps:

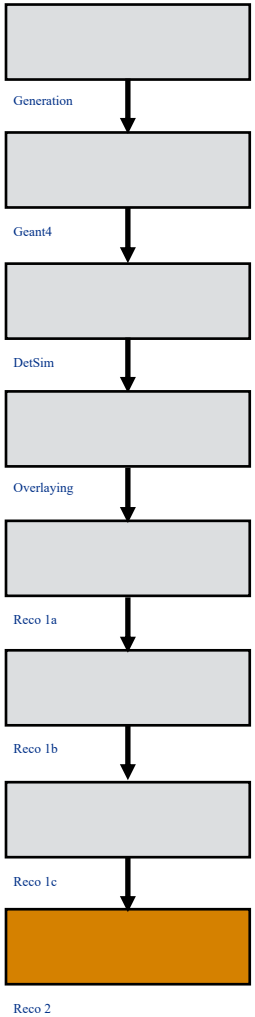
- 1a - Regular stage1 reco as done for data events, this stage uses the data trigger for setting the detector clock.

- 1b - Backtracking. This stage uses the MC trigger for setting the detector clock.

- 1c - Optical filtering. This stage uses again the data trigger for setting the detector clock.

Overlay Production

Reco2



Reco2 is similar to the Data Reco2 stage

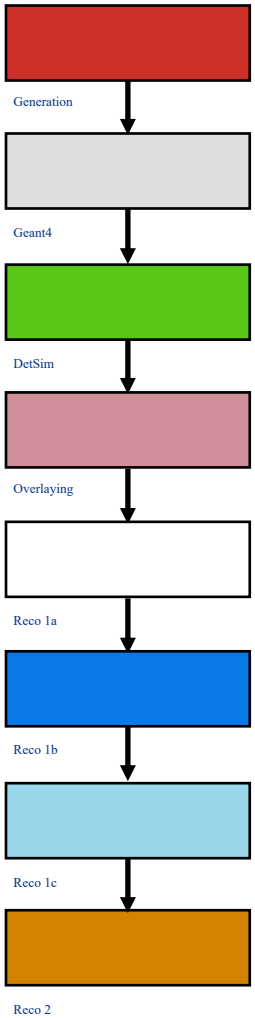
But also includes inline calibration (thank you Brandon!)

Applying:

$$K_{data}(x, y, z, t)$$

To all hits in all tracks (MC and data parts of the overlay)

Overlay Production to Validation



Full Documentation available here

Overlay - Mass Production

We wish to start overlay mass production at mid-March

Should take into considerations:

- Aiming for a factor of 10 more statistics of simulated events wrt data.
- We can overlay a signal on top on each EXT unbiased event and normalised using POT calculation
- We wish to compare data to overlay based on EXT unbiased data from the same Run period, and possibly same time of year from that period.
- Have different background for different signals such as ν_μ, ν_e, γ
- Input files have varying size, while a production job is for 50 events.
- It takes 7.5 hours on average to generate 50 events.

Overlay - Production

Run Period	POTs in data	equivalent events	available EXT Unbiased events	Factor in statistics EXT Unbiased / data
I	1.81E+20	1.64E+05	2.5E+06	15.2
II	3.05E+20	2.76E+05	4.8E+06	17.3
III	2.67E+20	2.42E+05	3.2E+06	13.2
IV	1.33E+20	1.20E+05	1.5E+06	12.4

We cannot afford to be wasteful with events and production time

Overlay - Production - Possible approaches

Since some ν interaction occupy small space:

one can divide one EXT unbiased events into many small ones.

POT count

Specify how we're doing that