DepoFluxWriter

Converting Drifted Depos to SimChannel IDEs

Brett Viren

August 17, 2023

1 Overview

The `DepoFluxWriter` converts a WCT `IDepoSet` received from a WCT flow graph edge to `IDE` in a `vector` of `SimChannel` which then is written to the `art::Event`.

Note, `DepoFluxWriter` supersedes `DepoSetSimChannelSink` and `SimChannelSink` both of which should be considered obsolete. They may be removed in the future. See the end of this document for information on migrating your configuration to `DepoFluxWriter`.

2 Interpreting the output

Each `SimChannel IDE` produced by `DepoFluxWriter` represents a subset of the "true" flux of ionization electrons through a particular **measurement plane** (MP) that is perpendicular to the nominal drift direction. The MP is identified with the **collection plane** (CP) by default (but see below).

At the MP, the flux of electrons is binned in drift time and spatially in the MP. In time, the flux is integrated over small sample periods ("ticks"). In space, the flux is integrated over the region near all electrodes that are connected to a common electronics channel. A region is taken to run the length of an electrode (wire or strip) and extend on either side of its center line by one half pitch.

In practice, each input depo is assumed to be the result of applying the WCT `Drifter` to a depo that was originally produced from a `SimEnergyDeposit`. The `SimEnergyDeposit` represents a small step of an ionizing track, typically made by Geant4. The `Drifter` transports input depos to a **response plane** (RP). In the full WCT simulation, the pile up of depos at the RP become input to the detector response simulation (`DepoTransform`). They may also be sent in to a `DepoFluxWriter`. The `DepoFluxWriter` further transports them to the MP and writes them out as collections of `SimChannel IDE`.

3 Nominal time

The **nominal time** is measured at the MP - again the collection plane by default. The time stored in the `IDE` is the `tdc` which represents the number of ticks from the start of an acceptance window to the tick that contains the nominal time.

In practice, the nominal time is the sum of the following times:

- absolute time at which tracking begins.
- time to create a given track.

- time to reach a given track step leading to a depo.
- time to drift that depo to the RP.
- time to transport that depo to the MP.

Thus, the **nominal time** measures when a portion of a depo passes through the **measurement plane** This time is absolute and correct given its definition and should not require any (in)corrections.

4 Acceptance window

The acceptance window governs the origin for the `tdc` count and which portions of depos will be converted to `IDE` contributions to `SimChannel`. It is defined in **nominal time**.

This means it must bracket the expected arrival times for all depos that you wish to convert. It must take into account any absolute time that is introduced upstream of `DepoFluxWriter`.

For example, if Geant4 is given an "event start time" of T0 and assuming interaction and tracking times are neglected, one may pick a window starting from T0 with a duration of one maximum drift time. This will exactly capture depos in the drift volume.

The acceptance window is configured using:

`tick` The sample time over which to integrate depo flux into time bins.

`window_start` The start of the acceptance window in **nominal time** measured at the MR.

`window_duration` The duration of the acceptance window.

The actual acceptance window will truncate `window_duration` so that it is an integral number of `tick`.

5 Incorrecting time offsets

To provide users with questionable flexibility, `DepoFluxWriter` accepts two types of arbitrary time offsets. These are only applied in calculating a `tdc` value and are not considered when applying the **acceptance window**. The configuration parameters are:

`reference_time` an absolute time **subtracted** from the **nominal time**. Default is 0.

`time_offsets` a 3-array providing a relative time **added** to the **nominal time** on a per-plane basis. Default is empty.

6 Other configuration

6.1 Additional smearing

`DepoFluxWriter` can apply additional smearing to depos independently in the longitudinal (time) and the transverse direction of each plane with the following options:

`smear_long` a unitless, scalar value giving the number of ticks over which to smear. It forms a second Gaussian sigma that is added in quadrature with the original longitudinal sigma of the depo.

`smear_tran` is a unitless, scalar or 3-array giving number of pitches over which to smear. If scalar, the same smearing is applied regardless of the plane.

6.2 Track IDs

By default, `DepoFluxWriter` will transfer the `IDepo::id()` to the `TrackIDE::trackID`. If the following option is given, this behavior changes:

`sed_label` an `art::Event` label at which a `vector<SimEnergyDeposit>` is expected.

When this label is set, then `DepoFluxWriter` will assume the `IDepo::id()` holds an index into the vector at the `art::Event` label given by `sed_label`. It will retrieve the `SimEnergyDeposit` at that index and pass both the `trackID` and `origTrackID` to the `TrackIDE`.

Note, this requires **collusion** with the creator of the original `IDepoSet` that was sent through WCT `Drifter`, the drifted version of which is given to the `DepoFluxWriter`. That is the `prior()` of each `IDepo` input to the `DepoFluxWriter` must be propertly produced. One creator that knows how to collude is `SimDepoSetSource` and it does so with this option:

`id_is_track` If **false**, `IDepo::id()` stores an **index** into `SimEnergyDeposit` vector to the element from which the `IDepo` was made. If **true** the `IDepo::id()` stores the `SimEnergyDeposit::TrackID()` as historically done. Default is **true**.

6.3 Expert debugging

`DepoFluxWriter` and `SimDepoSetSource` accept an expert debugging option:

`debug_file` name of a file into which text data is dumped.

This produces copious output, one line for each depo. By producing this data from both components one may validate that the vectors of `SimEnergyDeposit` and the `IDepoSet` retain correspondence between calls of each component.

The data is dumped only if `sed_label` is set on `DepoFluxWriter` and `id_is_track` is true in `SimDepoSetSource`.

7 Tests

In a suitable environment (see for example wire-cell-containers)

```
bats mrb/srcs/larwirecell/larwirecell/tests/test_depofluxwriter.bats
```

The test produces some resultf files which are deleted by `bats` by default. To save them run like:

```
WCT_BATS_TMPDIR=$(pwd)/tmp \
bats mrb/srcs/larwirecell/larwirecell/tests/test_depofluxwriter.bats
ls -l tmp/*.{root,pdf}
```

8 Obsolete

The `SimChannelSink` and `DepoSetSimChannelSink` are obsolete, may give wrong results and should not be used. This section gives a guide on how to transition their configuration to that of `DepoFluxWriter`.

`g4_ref_time` consider `reference_time`.

`*_time_offset` supply `time_offsets` array.

`*_to_rp` these option are not available though `time_offset_{u,v,w}` can be used. The distance from CP to RP is provided by the `field_response` data.

`start_time` this is nearly equivalent to `window_start`.

`readout_time` this is nearly equivalent to `window_duration`.

`drift_speed` this is provided by the `field_response` data.

`use_extra_sigma` use `smear_long` and `smear_tran`, see below

In the obsolete components, the `use_extra_sigma` enabled hard-coded additional smearing. For posterity, the hard-code is below.

```
// longitudinal
int nrebin = 1;
double time_slice_width = nrebin * m_speed * m_tick; // units::mm
double add_sigma_L =
    1.428249 * time_slice_width / nrebin / (m_tick / units::us); // units::mm
sigma_L =
    sqrt(pow(depo->extent_long(), 2) + pow(add_sigma_L, 2)); // / time_slice_width;

// transverse
double add_sigma_T = wbins.binsize();
if (iplane == 0)
    add_sigma_T *= (0.402993 * 0.3);
else if (iplane == 1)
    add_sigma_T *= (0.402993 * 0.5);
else if (iplane == 2)
    add_sigma_T *= (0.188060 * 0.2);

sigma_T = sqrt(pow(depo->extent_tran(), 2) + pow(add_sigma_T, 2)); // / wbins.binsize();
```