TPC Event Overlay Code Manual

Andreas Loeschcke Centeno

30th of August 2021

Contents

1 Introduction

The event overlay code was written to visualize the event overlap in the TPC volume. Since `basf2`only simulates single events, this was not directly possible in the framework but was instead done in standalone code.

The code is based on reading files containing single events and manually place the data to a new root file. This makes the output file unusable in `basf2`since this method only deals with the data tree but not the persistent tree. The code is not well-maintained and there are probably some memory leaks. Ideally the event overlay is moved to `basf2`since it is not sustainable to have the code as standalone code. If this turns out to be impossible the next best alternative would be to find a way to create a valid persistent tree such that only the overlay is done in standalone code but not the following modules like the digitizer and background rejection. In the current version these modules are re-implemented in the overlay code such that changes would have to be maintained in two places. It is also in the background rejection module where I suspect the memory leaks to be located.

2 Overlay Code

The code is executed via the command `root -l -b -q create_overlay.cc`.

In the very beginning the struct hit for saving the hits is defined. It essentially serves as replacement for the digits in `basf2`.

Then global variables for the TPC readout geometry are defined. The units are given in the units of the output file which are cm for lengths and s for times. Additionally, some parameters for the background rejection are defined.

In the main function the vector containing the hits is declared. Then the hit map containing the pixel ID and the index of the hit in the hit vector is declared. This is later used for applying the effect of pixel dead time. For the beam background additional hit maps are declared, because else the memory usage becomes too large. The hit maps for the beam background are not used for pixel dead time. While this might help slightly with the amount of hit from beam background it is not expected to be major source of loss and increases computing time quite drastically. It is important however, that for all event types the same hit vector is used.

Afterwards the function for reading the event files and placing them in the TPC volume are called. The function takes as arguments:

- **rate**: number of events in a time window of 4 TPC volumes (for overlay in signal volume only 2 TPC volumes are needed for accurate event numbers, but for visualization for the volumes adjacent to the signal volume more events are needed, such that the adjacent volumes don't look too empty)
- **infilename**: name including location of input file containing events to be placed in overlay
- **outfilename**: name of the output file
- **evtID**: eventID for later analysis purpose. Starts at 1 for $\Upsilon(4S)$ and counts up.
- **hits**: vector containing the hits
- **HitMap**: hitmap for invalidating hits based on pixel dead time
- **origindex**: *optional*: for placing additional $\Upsilon(4S)$ events in the overlay, the index of the original $\Upsilon(4S)$ in the input file is passed such that this event is not placed twice.

Inside the readfile() function

The signal $\Upsilon(4S)$ is placed with a separate function call where **origindex** is set to -2. Receiving a call with -2 a **origindex** causes the function to place exaclty one event at time t_0 into the volume.

Else the number is drawn from a Poisson distribution with the **rate** as mean value.

From the input file the SimHits are read. Some variables are declared in which the desired information is later stored. Transfusion coefficients are defined here as well (maybe it makes sense to define as global variables since they are fixed for all events).

Then, based on the number from the Poisson distribution the indices for the events from the input file for this event type are determined. In order to avoid including events multiple times, this is done in a loop where previous indices are stored and compared. The principle is the following:

- a random integer from the number of entries in the input file is drawn
- the vector containing already drawn indices is checked if there is a double
- if not the index is added to the vector
- if it is a new random index is drawn

This is not the most efficient way of doing this, but it gets the job done. Additionally, since this can in principle loop forever, there is a **failsave** implemented, such that after a certain number of tries the index is added anyway. This in only relevant for beam background since there the rate is very large and the input are not infinitely large.

After the indices for the events from the input have been determined, the coordinates and other properties of the hits in the event are calculated. The first step is to determine the time t of the event, i.e., where the event is placed in the overlay procedure. For this purpose, a random timestamp is drawn which represents the index of the bunch crossing with respect to t_0. Based on a calculation using the full design luminosity of the accelerator and the drift velocity in the TPC, there are 7454 bunch crossings during one TPC volume worth of drift. The timestamp is then converted into a z-coordinate of the hit within this coordinate system where t_0 also marks the origin for the z-coordinate. This means that events with a negative bunch crossing index, also have a negative z-coordinate which indicates that this event took place before the signal event.

Before this z-coordinate is applied to the hits of the event, the drift length for each hit has to be determined in order to correctly apply the modeled diffusion. This is done by reading the original coordinates of the hit and converting it to a drift length. Then the Gaussian smearing is applied.

Afterwards follows the calculation of the cellID of the hits on the readout. This is part of the beam background rejection by Christian Wessel. From this also the "reconstructed" coordinate of the hit is calculated.

Then, the properties of the hit are stored in the vector containing the pre-defined struct. In the end, the hitmap is filled with all the hits from the event.

Back in main() function after readfile() calls

After readfile() has been called for all event types, the background rejection functions are called. These follow exactly the form of the `basf2`files and are not further explained here. In the end, the output file is created and written.

List of Figures

List of Tables