FTPC SlowSimulator

Frank Simon
Max–Planck–Institut für Physik, München, Germany
fsimon@mppmu.mpg.de

September 13, 2002

Contents

1 The SlowSimulator **1**

1.1 Program Structure 1

1.1.1 StFtpcSlowSimulator 2

1.1.2 StFtpcSlowSimCluster 2

1.1.3 StFtpcSlowSimReadout 2

1.1.4 StFtpcRawWriter 3

1.2 Running the Program 3

1.2.1 The SlowSimulator in the STAR Reconstruction Chain 3

1.2.2 Using a local Database 3

1.2.3 Creating Input Files 4

2 The AssociationMaker **6**

2.1 AssociationMaker Program Structure 6

2.1.1 StAssociationMaker 6

2.1.2 StMcAnalysisMaker 7

1 The SlowSimulator

1.1 Program Structure

The FTPC SlowSimulator consists of several classes that perform specialized tasks. The main class that is called by the STAR reconstruction chain is `StFtpcSlowSimMaker`. This class follows the structure of STAR Maker classes and features initialization routines that

initialize parameters of the SlowSimulator at the beginning of the run and for each event. The routine `StFtpcSlowSimMaker::Make` initializes all necessary data and database readers and writers and then starts the simulation program `StFtpcSlowSimulator::Simulate` itself.

1.1.1 StFtpcSlowSimulator

This class contains the main part of the simulation program, which is executed by calling the subroutine `StFtpcSlowSimulator::Simulate`. This routine is called once per event and basically consists of one big loop that loops over all hits (read from GEANT) in the current event. This offers the possibility to add smearing functions to smear out the input coordinates to simulate the effect of reduced spatial resolution or to randomly eliminate hits to simulate the influence of additional inefficiencies. The GEANT information is transformed into FTPC coordinates (r, φ and row instead of x, y, z) and together with the simulated charge it is passed through simulation subroutines, which are wrapped in new classes and are called from `StFtpcSlowSimulator::Simulate` sequentially. It is important to note that throughout the whole simulation chain, all calculations are performed in global coordinates which are equal for the East and West FTPC. The specific distinctions are only made in the last step after the loop over all hits, in `StFtpcRawWriter`, where the simulated pad responses are written into the output tables.

1.1.2 StFtpcSlowSimCluster

This class, with its main routine `StFtpcSlowSimCluster::DriftDiffuse` simulates the evolution of the charge cloud as it drifts from its creation point to the readout chambers. It takes the field-dependent drift velocity, the lorentz angle due to $\vec{E} \times \vec{B}$ effects as well as diffusion and absorption into account.

1.1.3 StFtpcSlowSimReadout

The different subroutines of this class, called by `StFtpcSlowSimulator::Simulate` simulate different stages of the readout process. `StFtpcSlowSimReadout::Avalanche` calculates the electron multiplication in the wire chambers of the readout modules, `StFtpcSlowSimReadout::PadResponse` simulates the response of the readout pads and `StFtpcSlowSimReadout::ShaperResponse` simulates the influence of the shaper stage on the front end electronics boards. `StFtpcSlowSimReadout::PadResponse` digitizes the pad data by using the known relations between charge and ADC values. As new features at this stage, the ADC amplitude is corrected for realistic gain settings on the readout chamber to get consistency between simulated and measured amplitudes, and a Gaussian distributed random noise is added to the simulated data.

1.1.4 StFtpcRawWriter

The subroutine `StFtpcRawWriter::WriteArray` builds the complete DAQ sequences from the digitized ADC values. It takes the ASIC parameters of the readout electronics into account and corrects for the different mapping in the East and West FTPC. The finished sequences are written to the DAQ tables that are used in the STAR reconstruction chain.

1.2 Running the Program

The successful running of a FTPC simulation requires two basic ingredients, first the correct running of the SlowSimulator in the STAR reconstruction chain, and second the creation of compatible input data files.

1.2.1 The SlowSimulator in the STAR Reconstruction Chain

The FTPC SlowSimulator is designed to run within the STAR reconstruction framework, and is executed from the `bfc.C` macro by specifying a certain set of options. One example is

`.x bfc.C(1, "C2001 fss big GeantOut",` *filename*`)`

where *filename* is the input file containing simulated raw data (see next subsection). The output of the simulator is automatically passed to the FTPC ClusterFinder and to the tracker, in the end resulting in fully reconstructed data. In addition to the usual `event.root` file that contains the reconstructed data, a `geant.root` file that permits to match reconstructed data with the original GEANT input is written out. This output file is needed to analyze the results from the simulation, i.e. to extract efficiencies and momentum resolution by using the AssociationMaker.

1.2.2 Using a local Database

To use a local database, for example to include dead pads in the ClusterFinder or to change the gas parameters for the SlowSimulator, small changes in the code have to be made. The local database has to be copied to the directory `StarDb/ftpc`. The database access has to be changed in the `Init()` routine of the corresponding maker. The change is from:

```
m_gas        = (St_ftpcGas *)dblocal_calibrations("ftpcGas");
```

to

```
m_gas        = (St_ftpcGas *)local("ftpcGas");
```

1.2.3 Creating Input Files

The format used for the input files is the GEANT output format `*.fzd`. Files in this format need no further processing, they can be directly used in the reconstruction chain. This is usually the case for HIJING files, which have already been processed by GEANT.

In the case of using self-made special events, the `fzd` files have to be created by running GEANT. This is done from the Star Analysis Framework (`staf`) which is a FORTRAN - based environment. To run GEANT, standardized macros (`*.kumac`) are used. The input to GEANT is a text file in either the old (simple) or new (more flexible) format. Note that running GEANT over files with high multiplicity events takes considerable amount of computing time, especially with full physics switched on. The result of such a GEANT run is a `fzd` file that can be used in the simulator.

An example for a working GEANT macro is the following:

```
MACRO test nevent=1
* generate a standard geometry with a drawing
* and a keyword explanation:

debug on

detp geometry year2001 Field=5 Phys_off Split_off
make geometry

* create 4 pion tracks

swit 2 3
swit 4 3

gfile o 4tracks_tpc.fzd
user/input txold 4tracks_tpc.txold

rung 11 0

trig [nevent]

* exit
return
```

It creates a fzd file named `4tracks_tpc.fzd` from the input text file `4tracks_tpc.txold` by running the first event from the text file through GEANT. The GEANT settings use the year2001 setup with full positive field (5 kGauss). Physics effects are switched off.

Macros that start from scratch and create HIJING events and run them through the GEANT detector simulation to write out fzd files are more complicated. A working example is:

```
MACRO hijev nevent=100 file=hijing run=1
application data hijev.inp
' ================================================== '
' =====            Hijing Control file            ===== '
' ================================================== '
' Run number                        '  1
' Event number                      '  0
' Generator number                  '  31
' Frame/Energy                      '  'CMS'  200.
' Projectile  type/A/Z              '  'A'  197 79
' Target      type/Z/Z              '  'A'  197 79
' Impact parameter min/max (fm)                '  0.   3.
' Jet quenching (1=yes/0=no)                   '  0
' Hard scattering/pt jet (0/1, -thr)          '  0   -2.25
' Max # jets per nucleon (D=10)               '  10
' ihpr2(11), ihpr2(12)-pi0,k0,D,L,...decayoff '  1   1
' ihpr2(21)- keep daughters, ihpr2(18)        '  1   0
' set C/B production(C=1.5,B=5.36)            '  1.5
hijev.inp
* ======================================================
*                        GSTAR setup
detp  geometry year2001 Field=5 Phys_off Split_off
RNDM  $pid [run]
vsig  0.01  18.
ghist [file].his
gstat time size mult stak
*$
make  hij
make  geometry
make  gstar
make  control
*                        set a primitive dataset structure
mkdir evgen
cd    evgen
tdm/newtable particle particle 40000
cd ..
*                        I/O setup here
 user/input    u  evgen/particle.staf
 gfile         o  [file][run].fzd
```

```
*                        run event loop
do i = 1, [nevent]
  mod/call hijjet evgen/particle
  more            evgen/particle
  trig 1
enddo
*
exit
return
```

2 The AssociationMaker

The AssociationMaker is used to compare the reconstructed information with the original simulation data. For a detailed description of the program refer to the appropriate documentation. Here, a short overview of the relevant features is given.

2.1 AssociationMaker Program Structure

The AssociationMaker is run from a ROOT script, and it consists of two separate makers, the StAssociationMaker and the StMcAnalysisMaker.

2.1.1 StAssociationMaker

The StAssociationMaker is the program that does the association of reconstructed hits to MC hits, and the association of reconstructed tracks and MC tracks. For this association, certain parameters such as the maximum distance of points and the minimum number of common hits an a track have to be specified. This is done by the macro calling the AssociationMaker. The associated hits and track are written into multi-maps that are used in the later analysis.

A known problem of the AssociationMaker is that it only associates FTPC hits with $y > 0$. This is due to the sorting of the FTPC hits in their container and can be solved by removing the speed optimization that relies on the correct order of hits. This is done by changing

```
StMcFtpcHitIterator ftpcHitSeed
= find_if (mcFtpcHitColl->plane(iPlane)->hits().begin(),
  mcFtpcHitColl->plane(iPlane)->hits().end(),
  ftpcComp);
```

to

```
StMcFtpcHitIterator ftpcHitSeed
= mcFtpcHitColl->plane(iPlane)->hits().begin();
```

thus removing the `find_if` function that needs correct sorting of the hits.

2.1.2 StMcAnalysisMaker

The McAnalysisMaker is a program that has to be adapted to the specific requirements of the current analysis. It uses the results of the AssociationMaker, which is called from within the McAnalysisMaker, and evaluates them for further use. A common procedure is to create a tree that is filled with the difference between reconstructed and associated MC hits, or with the momentum difference of reconstructed and MC tracks, to evaluate spatial and momentum resolution. Here, it has to be kept in mind that the Association-Maker, dependent on the cuts, can associate several MC hits a to given reconstructed hit, and correspondingly can also associate several MC tracks to a reconstructed track. This requires further checks in order not to distort efficiency measurements.