The implementation of the RPC wall in the HGeant framework of HADES

Document prepared by A. Mangiarotti
LIP-Coimbra

October 9, 2013

1 The geometry

The geometry of the RPC is implemented with the usual HADES extension of Geant allowing all parameters to be stored in an external file (by convention with the ".geo" extension). The RPC volumes are identified by their names all beginning with the letter "E" (this choice is hard coded in `hgeomrpc.cc`). The main features of the HADES interface can be summarised as follows.

- The volumes are specified by a unique name of 4 characters. If the name contains five characters, the last one is automatically interpreted as a Geant copy number referring to the volume identified by the first four (see `GEOM100` [1]).
- The implemented subset of the Geant shapes available for each volume is `BOX`, `PGON`, `PCON`, `TRAP`, `TRD1`, `TUBE`, `TUBS`, `CONE`, `CONS`, `SPHE`, `ELTU` (see `GEOM050` [1]). Beyond the geometrical parameters, the file format allows to specify the filling material, the mother in which the volume has to be placed, as well as the offset vector and a 3x3 rotation matrix (see `GEOM100` and `GEOM110` [1]). The offset vector and rotation matrix have to be specified in the coordinate system of the mother volume.
- All volumes are positioned with the `ONLY` option of Geant to improve computation speed of particle transport. Then each volume should not overlap with any other volume except its mother, which must fully contain the volume, and its daughters, which must be fully contained into the volume (see `GEOM001` and `GEOM110` [1]).
- The part of the volume tree relevant for the RPC wall has two upper levels. The `CAVE` is the mother volume (`BOX`) containing all the HADES spectrometer. Within the cave volume, $N_{\mathrm{sectors}} = 6$ volumes (each corresponding to one of the six triangular right prisms `PGON` composing a right hexagonal prism with a constant outer radius and a variable inner radius) are defined as `SEC1` $\cdots N_{\mathrm{sectors}}$ to hold a sector made of all detectors plus two half coils with their casing and support structure.

Only one sector, number 1 (i.e. the central upper one) is implemented and then copied to all others using the Geant volume copy option, which automatically extends the copy to all contained sub-volumes (see `GEOM001` [1]). One Geant volume shape is best suited for the implementation of the complete RPC geometry: the TRAP. *Note that the most general trapezoid available in Geant is* `GTRAP`. The `TRAP` is a trapezoid with two planar and parallel faces (which are trapezia) (see `GEOM050` [1]). In the HADES framework, this volume is specified by the coordinates of the eight corners counted clockwise starting at the lower left corner of the bottom plane (contrary to native

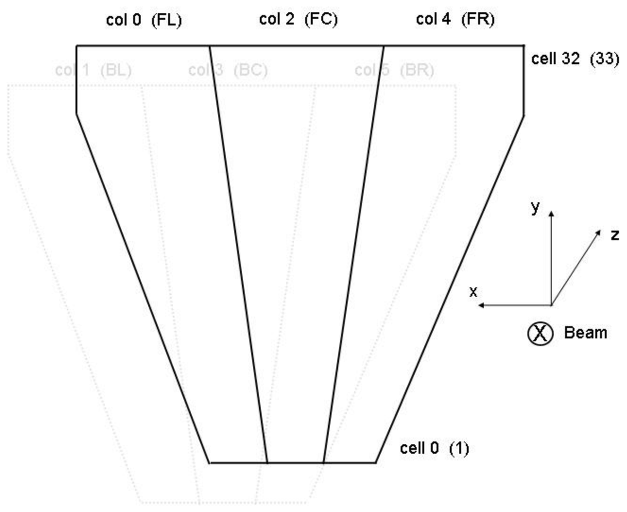

Figure 1: Column numbering convention. The picture refers to the Hydra numbering (C/C++) from 0 to 5. In Geant the columns are numbered from 1 to 6 (FORTRAN). The same difference appears for the cells and the gaps (not shown).

Geant, see `GEOM050` [1]). The intrinsic coordinate system of a `TRAP` is also different from the one in Geant. The y-axis points from the smaller side in x-direction to the larger one. A `TRAP` not rotated in a Box has the same intrinsic coordinate system as the `BOX`. In Geant the y- and x-axis point in the opposite directions.

The aluminium box containing all the cells of one sector has two identical internal cavities, which are filled with RPC gas, each containing one of the two layers of RPC cells. The shape of this cannot be described by a single trapezoid because of the corners (see also Fig. 1). The choice has been made to create the box out of $N_{\text{columns}} = 6$ trapezoidal volumes of aluminium, which will contain the columns, three for each layer. The front and back column meet in the middle of the central aluminium plate separating the two layers. The left and right columns have their corners cut by superimposing a triangular trapezoid made of air. The definition of each internal cavity for the left and right columns of the box also requires several trapezoids to be implemented. This is not only due to the shape but also to the presence of the RPC cells, which must be fully contained into only one of the internal sub-volumes. In the end, 9 internal trapezoids of 4 different shapes are necessary to create each of the two cavities, filled with RPC gas. The number of RPC detector cells is $N_{\text{cells}} = 32$ (note however that only the central column in the back actually has all the 32 cells, in the others the 32nd cell is missing). Each cell has $N_{\text{gaps}} = 4$ gas gaps delimited by $N_{\text{plates}} = 2$ glass plates and $N_{\text{electrodes}} = 3$ conducting electrodes. In detail, the tree of nested volumes is as follows (each volume at each level is fully contained into only one volume located at a higher position in the hierarchy, as required).

1. `EBOX[`$1 \cdots N_{\text{sectors}}$`]` is the topmost volume containing all the others. It is actually an active volume filled with air (`AIR_ACTIVE`), used to tag all crossing particles. This information is later used to distinguish hits produced by those particles from hits produced by their secondaries.

2. `ECO[`$1 \cdots N_{\text{columns}}$`]` are the six columns made of aluminium (`ALUMINIUM`). Columns 1, 3 and 5 are in the front when looked from the target, columns 2, 4 and 6 are the matching ones, respectively, in the back (see also Fig. 1). Placed inside `EBOX`.

3. `EV[`$1 \cdots N_{\text{columns}}$`][`$1 \cdots 5$`]` are the volumes necessary to cut the corners and internal cavities of the aluminium box. As mentioned, the middle columns have only one volume `EV31` and `EV41`

made of RPC gas (GAS_RPC_NOACTIVE). The left and right columns have one volume each EV15, EV55, EV25 and EV65 made of air (AIR) to cut the corners and other four volumes each made of RPC gas (GAS_RPC_NOACTIVE) to cut the internal cavity. Placed inside ECO.

4. ET[$1 \cdots N_{\text{columns}}$][$1 \cdots N_{\text{cells}}$] are the aluminium (ALUMINIUM) volumes containing each detector cell. The 32 cells are numbered from 1 to 9 and then using the letters from A to V. Placed inside EV.

5. EI[$1 \cdots N_{\text{columns}}$][$1 \cdots N_{\text{cells}}$] are the RPC gas (GAS_RPC_NOACTIVE) volumes necessary to define the internal cavities of the corresponding ET volumes so forming the shields of the detector cells. Placed inside ET.

6. EE[$1 \cdots N_{\text{columns}}$][$1 \cdots N_{\text{cells}}$][$1 \cdots N_{\text{electrodes}}$] are the aluminium (ALUMINIUM) electrodes of each cell and are implemented as copies of the first one (closest to the target). The four gaps are delimited by three conducting electrodes. Placed inside EI.

7. ER[$1 \cdots N_{\text{columns}}$][$1 \cdots N_{\text{cells}}$][$1 \cdots N_{\text{plates}}$] are the glass (GLASS) plates of each cell and are implemented as copies of the first one (closest to the target). The four gaps are delimited by two glass plates. Placed inside EI.

8. EG[$1 \cdots N_{\text{columns}}$][$1 \cdots N_{\text{cells}}$][$1 \cdots N_{\text{gaps}}$] are the four active gaps of each cell and are implemented as copies of the first one (closest to the target). They are filled with the ACTIVE version of the RPC gas (GAS_RPC), because hits produced in these volumes are stored by Geant. Placed inside EI.

To better understand the tree of nested volumes, here is an example of the meaningful content (i.e. down to level 8) of vectors NAMES (volume name) and NUMBER (volume copy number) from the common block /GCVOLU/ in the case of the third gas gap of cell O in column 4 of sector 2.

1	CAVE	1
2	SEC2	1
3	EBOX	2
4	ECO4	1
5	EV41	1
6	ET4O	1
7	EI4O	1
8	EG4O	3

Few comments are in order about the model used in Geant to describe the fluctuations of the energy lost in a thin gas layer (see PHYS332, PHYS333 and PHYS334 [1]) because they are relevant mostly for the RPC gap. To avoid double peak structures in the energy loss spectrum of low momentum particles, the Photoabsorption Ionization (PAI) model (selected with ISTRA=1) should be preferred to the Urbán model (selected with ISTRA=0). The PAI model has been introduced in Geant 3 to specifically simulate the fluctuation in thin layer and so its superior performances are to be expected in the present case of a very thin gas volume (see PHYS334 [1]). This recommendation is true irrespectively of whether the choice ILOSS=1 (i.e. with δ-ray generation) or ILOSS=2 (i.e. without δ-ray generation) is made and irrespectively of the value selected for the δ-ray emission cut DCUTE. Some tests have been performed, mostly with protons in the momentum range from 500 MeV/c to 1 Ge/c, to gauge the accuracy of the energy loss and energy loss fluctuation models. The results have been compared with those from HEED 1.01, a special code developed at CERN by I. Smirnov [2] to simulate energy deposition in gaseous detectors, also based on the PAI model. The standard RPC mixture has been implemented in HEED with the appropriate gas molecules, not the simple elemental composition as in Geant. The results are however very close to those of Geant, if the PAI model is used. The best accuracy versus performance optimisation is obtained by enabling ISTRA=1 for the RPC gas medium GAS_RPC and the active gas medium of the box surrounding the sector AIR_ACTIVE, but leave all other media with ISTRA=0, which is the recommended choice for solids or for gas volumes in general. This differential behaviour can be achieved by calling GSTPAR only for GAS_RPC and AIR_ACTIVE to explicitly set ISTRA=1. As explained in the Geant manual [1], in this way the new setting will take effect only for those two media. It is important that this call is done before calling GPYSI to initialise cross section and energy loss tables. A reasonable place to do this is inside UGINIT.

2 The hit structure

The Geant system employs a particle transport scheme based on a stack. The stack is initially loaded with all primary particles. If new particles are produced during transport by decay or discrete interactions with the detector material, they are also saved into the stack. The particles in the stack are processed in a LIFO order (see TRAK001 [1]). While this logic is very optimised for particle transport, it has a limitation if it is necessary to calculate the detector response. When a track crosses a detector cell, it cannot be known if other particles will also go through the same cell. The solution to this problem is the hit data structure provided by Geant (see HITS001 [1]). During particle transport the subroutine GUSTEP is called at each step to allow the user to take any appropriate action. In the HADES framework, GUSTEP calls in turn a special routine for each detector component. In the case of the RPC wall, the called subroutine is RPCSTEP. If the track is crossing the RPC gap, all necessary quantity to later calculate the response are saved as an element of the hit data structure associated with that specific detector cell. When all particles have been transported, Geant calls the user subroutine GUDIGI. In the HADES framework, GUDIGI calls in turn a special routine for each detector component. In the case of the RPC wall, the called subroutine is RPCDIGI. In RPCDIGI a loop is performed over all RPC cells and the response could be calculated including the description of the degradation due to several particles crossing the same active element. In reality, RPCDIGI simply passes the contents of all the hits to the Hydra framework, which saves them as a ROOT file. The actual response of the detector is applied only later, when the results of the simulation are processed through the Hydra framework.

To activate the hit data structure, a detector medium has to be declared as active: in the present case GAS_RPC. Then each active volume has to be associated to a Geant detector set, in the present case RPCG, and the hit data structure initialised with the following information (see HITS100 and HITS110 [1]).

1. Name of the set.
2. Name of the active volume.
3. Minimal list of the other volumes in the tree whose copy numbers are necessary to uniquely identify the volume. In the RPC case, they are the copy number of EBOX or sector number, and the copy number of the active volume itself EG or gap number. To save memory space, the copy numbers are packed using a number of bits that must also be specified.
4. A user defined detector type (an unpacked integer). In HADES each detector sub-system has a different offset. The RPC one is 500 and the detector type is formed as $500 + 35 * (n_{\text{column}} - 1) + (n_{\text{cell}} - 1)$ (hard coded in hgeomrpchit.cc).

5. The structure of the hit intended as the number of variables, all assumed floats, their mnemonic names and the packing information: number of bits NB(I), origin ORIG(I) and multiplicative factor FACT(I) where I is the order number of the hit variable. The packing is performed converting the float HIT(I) into an integer VAL(I) according to the relation VAL(I) = (HIT(I) + ORIG(I)) $*$ FACT(I) and then truncating VAL(I) to the prescribed number of bits NB(I) (see `HITS110` [1]). The minimum and maximum allowed values for VAL(I) are hence 0 and $2^{\mathrm{NB(I)}} - 1$, respectively (this information is not given in the manual [1], instead see the source code of subroutine `GSAHIT`). A special case is NB = 0 equivalent to NB = 32 where the transformation VAL(I) = (HIT(I) + ORIG(I)) $*$ FACT(I) is used with no packing, resulting in a minimum and maximum allowed values for VAL(I) of 0 and $2^{31} - 1$, respectively.

Again the HADES framework takes care of automatically introducing all this information into Geant by calling the appropriate routines with the necessary parameters starting from a single external file (by convention with the ".hit" extension). All RPC detectors belong to the same Geant set, whose name `RPCG` is defined in the external file. As an example of the automatically generated hit data structure, the case of the third gas gap of cell O in column 4 of sector 2 is considered explicitly. The copy numbers of volumes `ECO`, `EV`, `ET` and `EI` are actually not necessary, because they are always set to 1, but are automatically included by the framework.

```
SET RPCG   DETECTOR EG4O   NWHI=   1000   NWDI=   1000
           VOLUME EBOX   NBITSD=   2
           VOLUME ECO4   NBITSD=   1
           VOLUME EV41   NBITSD=   1
           VOLUME ET4O   NBITSD=   1
           VOLUME EI4O   NBITSD=   1
           VOLUME EG4O   NBITSD=   3
           HIT ELEMENT =X      NBITSH=  30  ORIG =  0.3400E+03  FACT =  0.1000E+05
           HIT ELEMENT =Y      NBITSH=  30  ORIG =  0.3400E+03  FACT =  0.1000E+05
           HIT ELEMENT =Z      NBITSH=  30  ORIG =  0.3400E+03  FACT =  0.1000E+05
           HIT ELEMENT =TOF    NBITSH=  30  ORIG =  0.0000E+00  FACT =  0.1000E+13
           HIT ELEMENT =ITRA   NBITSH=  16  ORIG =  0.0000E+00  FACT =  0.1000E+01
           HIT ELEMENT =ELOS   NBITSH=  30  ORIG =  0.0000E+00  FACT =  0.1000E+11
           HIT ELEMENT =IDET   NBITSH=  16  ORIG =  1.0000E+00  FACT =  0.1000E+01
           HIT ELEMENT =PMOM   NBITSH=  30  ORIG =  0.2000E+01  FACT =  0.1000E+05
           HIT ELEMENT =TLEN   NBITSH=  30  ORIG =  0.0000E+00  FACT =  0.1000E+05
           HIT ELEMENT =LLEN   NBITSH=  30  ORIG =  0.0000E+00  FACT =  0.1000E+05
           HIT ELEMENT =PDX    NBITSH=   0  ORIG =  0.1000E+01  FACT =  0.1000E+07
           HIT ELEMENT =PDY    NBITSH=   0  ORIG =  0.1000E+01  FACT =  0.1000E+07
           HIT ELEMENT =PDZ    NBITSH=   0  ORIG =  0.1000E+01  FACT =  0.1000E+07
```

Once initialised, the hit structure is filled in the `RPCSTEP` subroutine according to the following part of code, under the condition of a charged particle stopped in the active volume (ISTOP > 0, see `TRAK001` [1]) or exiting from it (INWVOL = 2, see `TRAK001` [1]). The call to `GMTOD` (see `GEOM320` [1]) is used to transform the hit position from the master reference frame (i.e. the reference frame of the mother volume `CAVE`) to the sector reference frame (i.e. the reference frame of the `EBOX` volume). The flags 1 and 2 simply indicate that a vector or the direction cosines have to be transformed, respectively (i.e. in the last case no translation vector is applied). The minus signs in the x and y coordinates of the hits and the direction cosines are necessary because of the mentioned difference between the reference frame of the `TRAP` volume in Geant and the natural choice for HADES, namely that for sector 1 (i.e. the central upper one) the z-axis points away from the target and the y-axis points in the upper vertical direction, the direction of the x-axis being fixed by right-handedness.

```
DO I=1,3
   XM(I) = VECT(I)
   PM(I) = VECT(I+3)
ENDDO

NLEVEL=3                           ! Geometry level for transforming to
                                   !             module ref. system (EBOX)
CALL GMTOD(PM,PD,2)                ! transform momentum cosines into module
                                   !                   ref. system (EBOX)
CALL GMTOD(XM,XD,1)                ! transform coordinates into module ref.
                                   !                       system (EBOX)

HITS(1)  = -XD(1)                  ! x of hit in detector coordinates
                                   !       (the minus sign is needed)
HITS(2)  = -XD(2)                  ! y of hit in detector coordinates
                                   !       (the minus sign is needed)
HITS(3)  = XD(3)                   ! z of hit in detector coordinates
HITS(4)  = TOFG                    ! ToF of hit
HITS(5)  = FLOAT(ITRA)             ! Geant track number
HITS(6)  = ELOS                    ! deposited energy
HITS(7)  = FLOAT(DETNR)            ! DETECTOR number
HITS(8)  = VECT(7)                 ! total momentum of particle
HITS(9)  = SLENG                   ! length of the current track
HITS(10) = SLENG - LENGTH_IN       ! local track length (used for energy
                                   !                  loss estimates)
HITS(11) = -PD(1)                  ! Px/P (the minus sign is needed)
HITS(12) = -PD(2)                  ! Py/P (the minus sign is needed)
HITS(13) = PD(3)                   ! Pz/P
```

The quantity ELOS is set to zero when the particle enters the gap (INWVOL = 1, see TRAK001 [1]) and then accumulates the energy lost per step as given by Geant in the variable DESTEP from the common block /GCTRACK/. The vector VECT from the common block /GCTRACK/ holds the particle spatial position (components $1-3$), the direction cosines of the particle momentum (components $4-6$) and the total particle momentum (component 7) at the current step. The calculated track length and time of flight to the current step are kept by Geant in the variables SLENG and TOFG, respectively, still from the common block /GCTRACK/. The track length inside the gap is calculated by setting LENGTH_IN = SLENG at the entrance (INWVOL = 1). The detector number DETNR is just the detector type minus 500, i.e. $35*(n_{\text{column}}-1)+(n_{\text{cell}}-1)$. Finally, ITRA is the Geant track number from the common block /GCKINE/. It is always defined for primary particles. For secondary particles, it may or may not be defined depending on the kind of behaviour selected with the key ISECO in the datacard. With the appropriate choice of ISECO (e.g. ISECO 2 0, but see hsecst.F and fillkine.F), it is possible to associate a track number to all secondary particles and save their initial vertex and kinematics in Hydra.

3 The Hydra categories for the simulation of the RPC detector

Hydra is the data analysis framework of HADES, it is designed in an object oriented way with C++ and based on ROOT. The main advantage of Hydra is to organise all the different data levels (i.e. raw, calibrated, hit and cluster) in a modular and consistent way. Depending on the task performed and the options specified, each data level is or is not streamed to a ROOT file.

For the RPC detector the objects corresponding to each level of data analysis are HRpcRaw, HRpcCal, HRpcHit and HRpcCluster for raw data, calibrated data, reconstructed hit and reconstructed clusters, respectively. Each of these levels is actually organised in a class called a category: a container of identical objects of the mentioned types with standardised access methods. The Hydra framework provides two fundamental types of categories

1. HLinearCategory is a container of objects indexed by a single integer: their position. It is basically a wrapper of a ROOT TClonesArray class. It is used in all cases when there is no predetermined maximum number of objects, e.g. for primary tracks in a Geant event (GeantKine category). The objects contained in the category can be accessed by the linear index giving their position in the TClonesArray or by iterating trough with the appropriate methods [1].

2. HMatrixCategory is a container of objects labelled by several indexes (i.e. in the case of the RPC detector these could be sector, column, cell for representing hits). This use naturally leads to a situation where the number of dimensions as well as the range of the indexes can be defined once for all instances of the same category. The implementation of HMatrixCategory still internally holds the objects in a TClonesArray of ROOT, but an additional lookup table is added to convert the several indexes to the single position index inside the TClonesArray. The objects contained in the category can be accessed by the linear index giving their position in the TClonesArray, by the set of multiple indexes or by iterating trough again with the appropriate methods.

The implementation strategy in Hydra is that all data levels for the RPC detector (e.g. HRpcRaw, HRpcCal, HRpcHit and HRpcCluster) are implemented as an HMatrixCategory because they can naturally benefit from an access scheme with several indexes. For internal conveniences in the Hydra software, this set of indexes is not the same at all levels, as described in the following table.

Level	data	simulation	n. of indexes	meaning
Raw	HRpcRaw	–	3	sector, column, cell
Simulation input	–	HGeantRpc	2	sector, max_trk
Calibrated	HRpcCal	HRpcCalSim	3	sector, column, cell
Hit	HRpcHit	HRpcHitSim	3	sector, column, cell
Cluster	HRpcCluster	HRpcCluster	2	sector, column*cell

In the table, max_trk is the maximum number of tracks per sector expected in the simulation and column*cell the combined index from the column and cell numbers. The categories are instantiated inside the class HRpcDetector.

The basic procedure to accommodate the simulation in this framework is to derive the objects corresponding to HRpcCalSim, HRpcHitSim and HRpcClusterSim from HRpcCal, HRpcHit and HRpcCluster, respectively, with the C++ inheritance construct. In this way, all internal variables declared as protected (not as private) and all public access methods of the parent classes are automatically present in the daughter classes as well. The daughter classes have then additional internal variables and access methods specific to the simulation case (e.g. track number). The raw level is not implemented in the simulation analysis chain, but the digitiser produces directly the calibrated level. The output from Geant is saved in a category of different objects, defined by the class HGeantRpc because the structure of HRpcRaw is too different and the inheritance construct is not convenient in this case. The main physical reason of this dichotomy being that when more than one track cross the same cell, while one single hit is anyhow generated in the raw data, in the simulation output the information relative to each track has to be stored. Again, HGeantRpc is implemented as an

[1]There is also a close analogy with the <vector> container class of the C++ Standard Template Library or the C++ Standard Library.

HMatrixCategory. The mentioned asymmetry between simulation and data explains also the different indexing scheme of HRpcRaw and HGeantRpc (see previous table). The hierarchy of the categories just described can be represented as follows, highlighting the parallel between data and simulation analysis. The task connecting one level to the next is also indicated.

$$\begin{tikzcd}
\texttt{HRpcRaw} \arrow[d, "\text{calibrater}\ (\texttt{hrpccalibrater.cc})"] & \texttt{HGeantRpc} \arrow[d, "\text{digitiser}\ (\texttt{hrpcdigitizer.cc})"] \\
\texttt{HRpcCal} \arrow[d, "\text{hit finder}\ (\texttt{hrpchitf.cc})"] & \texttt{HRpcCalSim} \arrow[d, "\text{hit finder}\ (\texttt{hrpchitf.cc})"] \\
\texttt{HRpcHit} \arrow[d, "\text{cluster finder}\ (\texttt{hrpcclusterf.cc})"] & \texttt{HRpcHitSim} \arrow[d, "\text{cluster finder}\ (\texttt{hrpcclusterf.cc})"] \\
\texttt{HRpcCluster} & \texttt{HRpcClusterSim}
\end{tikzcd}$$

The most important information (i.e. definition of the internal variables and the reading access methods) of the objects relative to each data level are given in the following subsections. For HGeantRpc there is only one class, for the other cases the parent class is described first and then the daughter.

3.1 HGeantRpc

The structure of the HGeantRpc object and its use have evolved with time. Before July of 2013, for each **gap** where energy has been deposited during particle transport, one such object is saved. Two more such objects are also saved corresponding to the entrance and the exit of the EBOX volume by each particle depositing some energy in at least one gap or producing any secondary particle depositing some energy in at least one gap.

```
private:
 Int_t     trackNumber;        // GEANT track number
 Float_t   trackLength;        // track length at the RPC gap.             [mm]
 Float_t   loctrackLength;     // local track length (inside the gap).     [mm]
 Float_t   eHit;               // energy deposited in the RPC gap.         [MeV]
 Float_t   xHit;               // x at the RPC gap, in module ref.
                               // system (EBOX).                           [mm]
 Float_t   yHit;               // y at the RPC gap, in module ref.
                               // system (EBOX).                           [mm]
 Float_t   zHit;               // z at the RPC gap, in module ref.
                               // system (EBOX).                           [mm]
 Float_t   tofHit;             // time of flight at the RPC gap.           [ns]
 Float_t   momHit;             // momentum at the RPC gap.                 [MeV/c]
 Float_t   thetaHit;           // polar angle at the RPC gap, in module ref.
                               // system (EBOX).                           [deg]
 Float_t   phiHit;             // azimuthal angle at the RPC gap, in module
                               // ref. system (EBOX).                      [deg]
 Short_t   detectorID;         // detector ID (codified sector/column/cell/gap)


public:

 // Functions getVariable
 inline Int_t  getSector(void) const;
 inline Int_t  getColumn(void) const;
 inline Int_t  getCell(void)   const;
 inline Int_t  getGap(void)    const;

 inline Int_t  getTrack(void)      {return trackNumber;}
 inline Int_t  getDetectorID(void) {return detectorID;}
        void   getHit(Float_t& axHit, Float_t& ayHit, Float_t& azHit,
                      Float_t& atofHit, Float_t& amomHit, Float_t& aeHit);
 // Optimized for digitizer.
        void   getHitDigi(Float_t& axHit, Float_t& atofHit, Float_t& amomHit,
                          Float_t& aloctracklength);
 //
        void   getIncidence(Float_t& athetaHit, Float_t& aphiHit);
        void   getTLength(Float_t& atracklength, Float_t& aloctracklength);
 inline Int_t  getNLocationIndex(void)   {return 4;}
 inline Int_t  getLocationIndex(Int_t i);
```

The information is redundant and not well organised. In fact, for the gaps of the same detector cell, some quantities like the track length from the target, the polar and azimuthal angles of incidence and the z coordinate do not change appreciably. Moreover, gaps belonging to the same detector cell, and hence not accessible independently in terms of readout signals, are scattered into several objects located at different positions inside the `HMatrixCategory`. For these two reasons, the `HGeantRpc` object has been reorganised to merge into a single object all the four gaps belonging to the same detector cell and purge unnecessary information. The new object structure must satisfy three requirements.

1. Allow to retrieve the average information for all the variables of the old version,
2. Allow to optionally store the relevant information for each gap independently,
3. Maintain backward compatibility.

The last condition is more severe than it might look. In fact, the `HGeantRpc` object is hold by the `HMatrixCategory`, which is a wrapper of a root `TClonesArray`. In turn, `TClonesArray` does not call the private streamer of `HGeantRpc`, but rather an internal optimised streamer. This automatic streamer to evolve between different versions of the same object must find variables with the same name, which will, by default, contain the same quantities when an older version of the object is read. It has been decided to keep unchanged the old object with the same variables and variable names and use it for a full **cell** (instead than for a single gap) by adding new variables to store the relevant information of the gaps. The new object structure is the following.

```
private:
 Int_t    trackNumber;         // GEANT track number
 Float_t trackLength;          // track length at the RPC gap/cell.       [mm]
 Float_t loctrackLength;       // local track length (inside the
                               // gap/cell).                              [mm]
 Float_t eHit;                 // energy deposited in the RPC gap/cell.  [MeV]
 Float_t xHit;                 // x at the RPC gap/cell, in module ref.
                               // system (EBOX).                          [mm]
 Float_t yHit;                 // y at the RPC gap/cell, in module ref.
                               // system (EBOX).                          [mm]
 Float_t zHit;                 // z at the RPC gap/cell, in module ref.
```

```
                              // system (EBOX).                             [mm]
 Float_t tofHit;              // time of flight at the RPC gap/cell.        [ns]
 Float_t momHit;              // momentum at the RPC gap/cell.              [MeV/c]
 Float_t thetaHit;            // polar angle at the RPC gap/cell, in
                              // module ref. system (EBOX).                 [deg]
 Float_t phiHit;              // azimuthal angle at the RPC gap/cell, in
                              // module ref. system (EBOX).                 [deg]
 Short_t detectorID;          // detector ID (codified
                              // sector/column/cell/gap info)
// New elements after Jan 2013.
 Float_t loctrackLength1;     // local track length (inside the gap).       [mm]
 Float_t eHit1;               // energy deposited in the RPC gap.           [MeV]
 Float_t xHit1;               // x at the RPC gap, in module ref.
                              // system (EBOX).                             [mm]
 Float_t yHit1;               // y at the RPC gap, in module ref.
                              // system (EBOX).                             [mm]
 Float_t momHit1;             // momentum at the RPC gap.                   [MeV/c]
 Float_t loctrackLength2;     // local track length (inside the gap).       [mm]
 Float_t eHit2;               // energy deposited in the RPC gap.           [MeV]
 Float_t xHit2;               // x at the RPC gap, in module ref.
                              // system (EBOX).                             [mm]
 Float_t yHit2;               // y at the RPC gap, in module ref.
                              // system (EBOX).                             [mm]
 Float_t momHit2;             // momentum at the RPC gap.                   [MeV/c]
 Float_t loctrackLength3;     // local track length (inside the gap).       [mm]
 Float_t eHit3;               // energy deposited in the RPC gap.           [MeV]
 Float_t xHit3;               // x at the RPC gap, in module ref.
                              // system (EBOX).                             [mm]
 Float_t yHit3;               // y at the RPC gap, in module ref.
                              // system (EBOX).                             [mm]
 Float_t momHit3;             // momentum at the RPC gap.                   [MeV/c]
 Short_t HGeantRpc_version;   // HGeantRpc class version when reading
                              // from file.
```

This structure gives the backward compatibility and the new wanted reorganisation of the information in a cell wise fashion. It has been adopted in the official version of Hydra in July of 2013.

This opportunity of changing the `HGeantRpc` object has also been taken to rationalise the access methods with the goal of providing a more modular and flexible interface to the information stored: in general only the quantities that are really necessary should be retrieved. This might led to probably small performance improvements in other parts of the code, like the digitiser, where the information is accessed frequently. Access methods referring to the "Gap" are meant to be used to get the quantities referring to each gap independently, while methods referring to the "Cell Average" are meant to be used to access the same quantities but averaged over the cell (only gaps with non zero energy deposition are including in the averaging). The old access methods have also been preserved to allow the old user code to work with the old version of `HGeantRpc`.

```
public:

// Functions getVariable
 inline Int_t   getSector(void) const;
 inline Int_t   getColumn(void) const;
 inline Int_t   getCell(void)   const;
 inline Int_t   getGap(void)    const;

 inline Int_t   getTrack(void)      {return trackNumber;};
 inline Int_t   getDetectorID(void) {return detectorID;};
 inline Int_t   getNLocationIndex(void) {return 4;};
 inline Int_t   getLocationIndex(Int_t i);
 inline Int_t   getVersion(void) {return HGeantRpc_version;};
//
//  OLD get functions for backward compatibility.
//
 void    getIncidence(Float_t& athetaHit, Float_t& aphiHit);
 void    getTLength(Float_t& atracklength, Float_t& aloctracklength);
 void    getHit(Float_t& axHit, Float_t& ayHit, Float_t& azHit,
                Float_t& atofHit, Float_t& amomHit, Float_t& aeHit);
//
// NEW GAP get functions.
//
// Various options are available to avoid retrieving unnecessary information.
 Float_t getlocTLengthGap(Int_t nGap);
 void    getGap(Int_t nGap, Float_t& axHit, Float_t& ayHit, Float_t& amomHit,
                Float_t& aeHit, Float_t& aloctrackLength);
 void    getGap(Int_t nGap, Float_t& axHit, Float_t& ayHit, Float_t& amomHit,
                Float_t& aeHit);
 void    getGap(Int_t nGap, Float_t& axHit, Float_t& ayHit, Float_t& amomHit);
//
// NEW HIT get functions.
//
 inline Float_t getTLengthHit(void) {return trackLength;};
 inline Float_t getZHit(void) {return zHit;};
 inline Float_t getTofHit(void) {return tofHit;};
// Various options are available to avoid retrieving unnecessary information.
 void   getHit(Float_t& axHit, Float_t& ayHit, Float_t& azHit, Float_t& atofHit,
               Float_t& amomHit, Float_t& aeHit, Float_t& aloctrackLength);
 void   getHit(Float_t& axHit, Float_t& ayHit, Float_t& azHit, Float_t& atofHit,
               Float_t& amomHit);
 void   getHit(Float_t& axHit, Float_t& ayHit, Float_t& azHit, Float_t& atofHit);
// Optimized for digitizer.
 void   getHitDigi(Float_t& axHit, Float_t& atofHit, Float_t& amomHit,
                   Float_t& aloctrackLength);
//
// NEW CELL get functions.
//
// Various options are available to avoid retrieving unnecessary information.
 void   getCellAverage(Float_t gap, Float_t& axHit, Float_t& ayHit, Float_t& azHit,
                       Float_t& atofHit, Float_t& amomHit, Float_t& aeHit,
                       Float_t& aloctrackLength);
 void   getCellAverage(Float_t& axHit, Float_t& ayHit, Float_t& azHit,
                       Float_t& atofHit, Float_t& amomHit, Float_t& aeHit);
 void   getCellAverage(Float_t& axHit, Float_t& ayHit, Float_t& azHit,
                       Float_t& atofHit, Float_t& amomHit);
 void   getCellAverage(Float_t& axHit, Float_t& ayHit, Float_t& azHit,
                       Float_t& atofHit);
```

```
// Optimized for digitizer.
 void    getCellAverageDigi(Float_t gap, Float_t& axHit, Float_t& atofHit,
                            Float_t& amomHit, Float_t& aloctrackLength);
```

3.2 HRpcCal and HRpcCalSim

The information about the logBit variable are not reproduced because, at the present stage, it is not relevant for the simulation.

PARENT

```
protected:

 Float_t rightTime;      //right leading time     [ns]
 Float_t leftTime;       //left leading time      [ns]
 Float_t rightTime2;     //2nd right leading time [ns]
 Float_t leftTime2;      //2nd left leading time  [ns]
 Float_t rightCharge;    //right charge           [pC]
 Float_t leftCharge;     //left  charge           [pC]
 Float_t rightCharge2;   //2nd right charge       [pC]
 Float_t leftCharge2;    //2nd left  charge       [pC]
 Short_t address;        //Geometrical address (sec, col, cell)


public:

 // Functions getVariable
 Float_t getRightTime()          { return rightTime;    }
 Float_t getLeftTime()           { return leftTime;     }
 Float_t getRightTime2()         { return rightTime2;   }
 Float_t getLeftTime2()          { return leftTime2;    }
 Float_t getRightCharge()        { return rightCharge;  }
 Float_t getLeftCharge()         { return leftCharge;   }
 Float_t getRightCharge2()       { return rightCharge2; }
 Float_t getLeftCharge2()        { return leftCharge2;  }
 Short_t getAddress()            { return address;      }

 Int_t   getNLocationIndex()     { return 3;}
 Int_t   getSector()            const  { return (address>>10)  & 7;  }
 Int_t   getColumn()            const  { return (address>>7)   & 7;  }
 Int_t   getCell()              const  { return  address       & 127;}
```

DAUGHTER

```
protected:

 Int_t RefL;            //Reference to the first mother contributing
                        //to the cell
 Int_t RefR;
 Int_t RefLDgtr;        //Reference to the first daughter contributing
                        //to the cell
 Int_t RefRDgtr;

 Int_t TrackL[10];      //Array of tracks of mothers at the RPC box
                        //for this cell
 Int_t TrackR[10];
 Int_t TrackLDgtr[10]; //Array of tracks of daughters
 Int_t TrackRDgtr[10];

 Int_t nTracksL;        //Number of daughter tracks at this cell
 Int_t nTracksR;

 Bool_t LisAtBox[10];   //Are they at box?
 Bool_t RisAtBox[10];


public:

 // Functions getVariable
 Int_t getRefL()                                  { return RefL;      }
 Int_t getRefR()                                  { return RefR;      }
 Int_t getRefLDgtr()                              { return RefLDgtr; }
 Int_t getRefRDgtr()                              { return RefRDgtr; }

 Int_t getTrackL()                const          { return TrackL[0];      }
 Int_t getTrackR()                const          { return TrackR[0];      }
 Int_t getTrackLDgtr()            const          { return TrackLDgtr[0]; }
 Int_t getTrackRDgtr()            const          { return TrackRDgtr[0]; }

 Bool_t getLisAtBox()             const          { return LisAtBox[0]; }
 Bool_t getRisAtBox()             const          { return RisAtBox[0]; }

 Int_t getNTracksL()                              { return nTracksL; }
 Int_t getNTracksR()                              { return nTracksR; }

 void  getTrackLArray(Int_t* trackLarray)
        {for(Int_t i=0;i<10;i++) trackLarray[i]      = TrackL[i];      }
 void  getTrackLDgtrArray(Int_t* trackLDgtrarray)
        {for(Int_t i=0;i<10;i++) trackLDgtrarray[i] = TrackLDgtr[i]; }
 void  getTrackRArray(Int_t* trackRarray)
        {for(Int_t i=0;i<10;i++) trackRarray[i]      = TrackR[i];      }
 void  getTrackRDgtrArray(Int_t* trackRDgtrarray)
        {for(Int_t i=0;i<10;i++) trackRDgtrarray[i] = TrackRDgtr[i]; }
 void  getLisAtBoxArray(Bool_t* LisAtBoxarray)
        {for(Int_t i=0;i<10;i++) LisAtBoxarray[i]    = LisAtBox[i];    }
 void  getRisAtBoxArray(Bool_t* RisAtBoxarray)
        {for(Int_t i=0;i<10;i++) RisAtBoxarray[i]    = RisAtBox[i];    }
```

3.3 HRpcHit and HRpcHitSim

The information about the logBit variable are not reproduced because, at the present stage, it is not relevant for the simulation.

PARENT

```
protected:

 Float_t tof;              //Time of flight [ns]
 Float_t charge;           //Charge [pC]
 Float_t xmod;             //X coordinate in Module system [mm]
 Float_t ymod;             //Y coordinate in Module system [mm]
 Float_t zmod;             //Z coordinate in Module system [mm]
 Float_t xsec;             //X coordinate in Sector system [mm]
 Float_t ysec;             //Y coordinate in Sector system [mm]
 Float_t zsec;             //Z coordinate in Sector system [mm]
 Float_t xlab;             //X coordinate in Lab system [mm]
 Float_t ylab;             //Y coordinate in Lab system [mm]
 Float_t zlab;             //Z coordinate in Lab system [mm]
 Float_t theta;            //Theta angle [degrees]
 Float_t phi;              //Phi angle [degrees]

 Bool_t  isInCell;         //Flag for cell outliers
                           //(false (zero) if xMod falls out of
                           //geometric bounds; true (one) otherwise)

 Float_t sigma_x;          //Sigma of x   [mm]
 Float_t sigma_y;          //Sigma of y   [mm]
 Float_t sigma_z;          //Sigma of z   [mm]
 Float_t sigma_tof;        //Sigma of tof [ps]

 Short_t address;          //Geometrical address (sec, col, cell)


public:

 // Functions getVariable
 Float_t getTof()              { return  tof;       }
 Float_t getCharge()           { return  charge;    }
 Float_t getXMod()             { return  xmod;      }
 Float_t getYMod()             { return  ymod;      }
 Float_t getZMod()             { return  zmod;      }
 Float_t getXRMS()             { return  sigma_x;   }
 Float_t getYRMS()             { return  sigma_y;   }
 Float_t getZRMS()             { return  sigma_z;   }
 Float_t getTOFRMS()           { return  sigma_tof; }
 Float_t getXSec()             { return  xsec;      }
 Float_t getYSec()             { return  ysec;      }
 Float_t getZSec()             { return  zsec;      }
 Float_t getTheta()            { return  theta;     }
 Float_t getPhi()              { return  phi;       }
 Bool_t  getInsideCellFlag()   { return  isInCell;  }

 void    getXYZLab(Float_t &x,Float_t &y,Float_t &z) {x=xlab;y=ylab;z=zlab;}

 Int_t   getNLocationIndex()   { return 3;          }
 Int_t    getSector()  const   { return (address>>10) & 7;  }
 Int_t    getColumn()  const   { return (address>>7)  & 7;  }
 Int_t    getCell()    const   { return  address      & 127; }
```

DAUGHTER

```
protected:

 Int_t RefL;           //Reference to the first mother contributing
                       //to the cell
 Int_t RefR;
 Int_t RefLDgtr;       //Reference to the first daughter contributing
                       //to the cell
 Int_t RefRDgtr;

 Int_t TrackL[10];     //Array of tracks of mothers at the RPC box
                       //for this cell
 Int_t TrackR[10];
 Int_t TrackLDgtr[10]; //Array of tracks of daughters
 Int_t TrackRDgtr[10];

 Int_t nTracksL;       //Number of daughter tracks at this cell
 Int_t nTracksR;

 Bool_t LisAtBox[10];  //Are they at box?
 Bool_t RisAtBox[10];


public:

 // Functions getVariable
 Int_t getRefL()                              { return RefL;     }
 Int_t getRefR()                              { return RefR;     }
 Int_t getRefLDgtr()                          { return RefLDgtr; }
 Int_t getRefRDgtr()                          { return RefRDgtr; }

 Int_t getTrackL()                const       { return TrackL[0];     }
 Int_t getTrackR()                const       { return TrackR[0];     }
 Int_t getTrackLDgtr()            const       { return TrackLDgtr[0]; }
 Int_t getTrackRDgtr()            const       { return TrackRDgtr[0]; }

 Bool_t getLisAtBox()             const       { return LisAtBox[0]; }
 Bool_t getRisAtBox()             const       { return RisAtBox[0]; }

 Int_t getNTracksL()                          { return nTracksL; }
 Int_t getNTracksR()                          { return nTracksR; }

 void  getTrackLArray(Int_t* trackLarray)
       {for(Int_t i=0;i<10;i++) trackLarray[i]     = TrackL[i];     }
 void  getTrackLDgtrArray(Int_t* trackLDgtrarray)
       {for(Int_t i=0;i<10;i++) trackLDgtrarray[i] = TrackLDgtr[i]; }
 void   getTrackRArray(Int_t* trackRarray)
         {for(Int_t  i=0;i<10;i++) trackRarray[i]       = TrackR[i];        }
 void   getTrackRDgtrArray(Int_t* trackRDgtrarray)
         {for(Int_t  i=0;i<10;i++) trackRDgtrarray[i] = TrackRDgtr[i]; }
 void   getLisAtBoxArray(Bool_t* LisAtBoxarray)
         {for(Int_t  i=0;i<10;i++) LisAtBoxarray[i]    = LisAtBox[i];     }
 void   getRisAtBoxArray(Bool_t* RisAtBoxarray)
         {for(Int_t  i=0;i<10;i++) RisAtBoxarray[i]    = RisAtBox[i];     }
```

3.4 HRpcCluster and HRpcClusterSim

PARENT

```
protected:
 Float_t tof;              //Time of flight [ns]
 Float_t charge;           //Charge [pC]
 Float_t xmod;             //X coordinate in Module system [mm]
 Float_t ymod;             //Y coordinate in Module system [mm]
 Float_t zmod;             //Z coordinate in Module system [mm]
 Float_t xsec;             //X coordinate in Sector system [mm]
 Float_t ysec;             //Y coordinate in Sector system [mm]
 Float_t zsec;             //Z coordinate in Sector system [mm]
 Float_t xlab;             //X coordinate in Lab system [mm]
 Float_t ylab;             //Y coordinate in Lab system [mm]
 Float_t zlab;             //Z coordinate in Lab system [mm]
 Float_t theta;            //Theta angle [degrees]
 Float_t phi;              //Phi angle [degrees]

 Float_t sigma_x;          //Sigma of x   [mm]
 Float_t sigma_y;          //Sigma of y   [mm]
 Float_t sigma_z;          //Sigma of z   [mm]
 Float_t sigma_tof;        //Sigma of tof [ps]

 Int_t   isInCell;         //Flag for cell outliers (xMod out of geometric
                           //cell bounds)
                           // If cluster type == 2
                           //   0 if both hits in the cluster are out of bounds
                           //   1 if both are in
                           //   2 if only first hit is in
                           //   3 if only second hit is in
                           //If cluster type ==1
                           //   0 if out of bounds
                           //   1 otherwise

 Short_t sector;           //Sector
 Short_t index;            //Linear index of "this" object in category

 Short_t detID1;           //Detector ID of the first  cell in the cluster
 Short_t detID2;           //Detector ID of the second cell in the cluster
```

```
public:

 // Functions getVariable
 Float_t getTof()             { return  tof;          }
 Float_t getCharge()          { return  charge;       }
 Float_t getXSec()            { return  xsec;         }
 Float_t getYSec()            { return  ysec;         }
 Float_t getZSec()            { return  zsec;         }
 Float_t getXMod()            { return  xmod;         }
 Float_t getYMod()            { return  ymod;         }
 Float_t getZMod()            { return  zmod;         }
 Float_t getTheta()           { return  theta;        }
 Float_t getPhi()             { return  phi;          }
 Float_t getXRMS()            { return  sigma_x;      }
 Float_t getYRMS()            { return  sigma_y;      }
 Float_t getZRMS()            { return  sigma_z;      }
 Float_t getTOFRMS()          { return  sigma_tof;    }
 Short_t getSector()          { return  sector;       }
 Short_t getIndex()           { return  index;        }
 Int_t   getClusterType()     { return  Int_t(type); }
 Int_t   getInsideCellFlag()  { return isInCell;     }

 Int_t   getSector1()  const  { if(detID1<0) return -1;
                                return (detID1>>9) & 7;  }
 Int_t   getColumn1()  const  { if(detID1<0) return -1;
                                return (detID1>>6) & 7;  }
 Int_t   getCell1()    const  { if(detID1<0) return -1;
                                return  detID1     & 63; }

 Int_t   getSector2()  const  { if(detID2<0) return -1;
                                return (detID2>>9) & 7;  }
 Int_t   getColumn2()  const  { if(detID2<0) return -1;
                                return (detID2>>6) & 7;  }
 Int_t   getCell2()    const  { if(detID2<0) return -1;
                                return  detID2     & 63; }

 void getXYZLab(Float_t &x,Float_t &y,Float_t &z)
                              {x=xlab;y=ylab;z=zlab;}
```

DAUGHTER

```
protected:
  Int_t TrackList[4];     //List of the four tracks contributing to the
                          //cluster, ordered as:
                          //UpLeft, UpRight, DownLeft, DownRight.
  Int_t RefList[4];       //List of the four references to the geantrpc
                          //objects contributing to the cluster, ordered as:
                          //UpLeft, UpRight, DownLeft, DownRight.
                          //Note: In case of cluster type = 1, the order is
                          //simply: Left, Right, -, -

  Bool_t isAtBoxList[4]; //Boolean list to indicate if a mother was found at
```

```
                            //the box or not.

  Int_t Track;              //Track preferred by the cluster finder (typically
                            //the one contributing more to the 4 possible
                            //times).

  Int_t nTracksAtBox;       //Number of different tracks at the RPC box that
                            //contribute to this cluster

  Int_t nTracksAtCells;     //Number of different tracks at the RPC cells that
                            //contribute to this cluster

  Bool_t isAtBox;           //Boolean to indicate if a mother was found at the
                            //box or not.


public:

 Int_t  howManyTracks()                  const  {return nTracksAtBox;}
 Int_t  howManyTracksAtCells()           const  {return nTracksAtCells;}

 // Functions getVariable
 Int_t  getTrack()                              {return Track;    }
 Bool_t getIsAtBox()                            {return isAtBox;  }

 void   getTrackList(Int_t* array)
                        {for(Int_t i=0;i<4;i++) array[i]=TrackList[i];}
 void   getRefList(Int_t* array)
                        {for(Int_t i=0;i<4;i++) array[i]=RefList[i];}
 void   getIsAtBoxList(Bool_t* array)
                        {for(Int_t i=0;i<4;i++) array[i]=isAtBoxList[i];}

 Bool_t isTrack(Int_t track);
 Bool_t isRef(Int_t ref);
```

4 The link between HGeant and Hydra

The connection between Geant and Hydra is performed in `RPCDIGI` which loops over all columns, cells and gaps and transfers the hit data into a common block `/RPCTUPLE/`, then it passes control to `fillrpc`, which is a C function bridging the gap with C++. The common block `/RPCTUPLE/` is mapped into C struct to allow access of the informations inside `fillrpc`. The common block `/RPCTUPLE/` is organised with a progressing hit number from one to NTRK and has the following structure.

```
       INTEGER NTRK
       INTEGER RPCTRK, RPCDET
       REAL RPCE, RPCX, RPCY, RPCZ, RPCTHETA, RPCPHI, RPCTOF, RPCMOM,
      &  RPCLEN, RPCLOCLEN

       COMMON /RPCTUPLE/
      &         NTRK,                 !current number of hits.
```

```
&        RPCTRK(MAXTRKRPC) ,       !GEANT track number.
&        RPCDET(MAXTRKRPC) ,       !detector ID (codified
                                   !sector/column/cell/gap)
&        RPCE(MAXTRKRPC) ,         !energy deposited in the RPC gap. [MeV]
&        RPCX(MAXTRKRPC) ,         !x at the RPC gap, in module ref.
                                   !system. [mm]
&        RPCY(MAXTRKRPC) ,         !y at the RPC gap, in module ref.
                                   !system. [mm]
&        RPCZ(MAXTRKRPC) ,         !z at the RPC gap, in module ref.
                                   !system. [mm]
&        RPCTHETA(MAXTRKRPC) ,     !theta angle of particle momentum
                                   !at the RPC gap.
&        RPCPHI(MAXTRKRPC) ,       !phi angle of particle momentum
                                   !at the RPC gap.
&        RPCTOF(MAXTRKRPC) ,       !time of flight at the RPC gap. [ns]
&        RPCMOM(MAXTRKRPC) ,       !momentum at the RPC gap. [MeV/c]
&        RPCLEN(MAXTRKRPC) ,       !track length at the RPC gap. [mm]
&        RPCLOCLEN(MAXTRKRPC)      !local track length (inside the gap). [mm]
```

Because the information is stored organised in a per gap fashion in /RPCTUPLE/ and in a per cell fashion in HGeantRpc, the conversion is performed inside fillrpc by grouping together the gaps crossed by the same track according to the physical cells. This internal reorganisation of the information is accomplished by using a C++ <vector> of structs of type celldat, addressed with a combined sector, column and cell number. The struct celldat internally holds a C++ <vector> of structs of type celltrack for each track crossing the cell. Finally, the struct of type celltrack stores the quantities referring to each gap. The variables which are not stored independently for each gap in HGeantRpc are averaged over the number of gaps with a non zero energy deposition. It should be noted that essentially the same code with similar struct types is present inside the digitiser to perform the same conversion from gap wise to a cell wise organisation of the information when the old version of HGeantRpc in use before June 2013 is found. The main difference is that in the latter case only the limited quantities necessary for the internal operation of the digitiser are considered, while in fillrpc all quantities present in HGeantRpc are treated.

Beyond looping over the hits, creating and filling the instances of HGeantRpc, the other important task performed in fillrpc is to remove uninteresting hits. This does not affect hits in the RPC gaps, but only hits in the EBOX volume by not storing those produced by tracks which directly or indirectly trough their secondaries do not produce any hit in any RPC gap. As a matter of fact, depending on the type of collision system and energy, this can reduce the dimension of space occupied on disk by up to $\approx 20\%$. The connection between the daughter tracks producing the hits and the parents crossing the EBOX volume is done resorting to the information in the HGeantKine container category (before it is itself purged by the call to finalizekine). The particles passing trough the RPC without producing hits are mostly neutrons generated in the primary interaction or by spallation from protons in the spectrometer material.

The loop over the sectors is done outside the loops over all columns, cells and gaps and results in independent calls to fillrpc for each sector. Beyond minimising the dimension of the common block /RPCTUPLE/, the separation into sectors speeds up the hit purging procedure just described.

As a technical detailed, it should also be mentioned that the currently used RPC geometry file (i.e. the one with extension ".geo") is not consistent with the axis and volume naming conventions of Fig. 1, which are however strictly followed in the part of Hydra handling the analysis of real data. Such inconsistency has the consequence that the right and left columns are interchanged in HGeant for both layers. The problem is remedied inside fillrpc by swapping back the lateral columns to their correct positions. In this way, all output files are fully consistent with the conventions of Fig. 1.

parameter name	model I	Au+Au	model II	Au+Au	units
fVprop	v_{prop}	177	v_{prop}	177	[mm/ns]
fS_x	σ_x	8	σ_x	8	[mm]
fS_time	σ_t	80	$\sigma_{t\,0}$	$4.38223E+01$	[ps]
fS1_time	no		$\sigma_{t\,1}$	$4.31246E+01$	[ps]
fS2_time	no		$\sigma_{t\,2}$	$6.87860E+00$	
fS3_time	no		$\sigma_{t\,3}$	$4.40930E-01$	
fT_off	t_{off}	63	t_{off}	63	[ps]
fQmean	q_{mean}	0.5	$q_{\mathrm{mean}\,0}$	$1.08095E+00$	[pC]
fQmean1	no		$q_{\mathrm{mean}\,1}$	$-2.13147E+00$	[pC]
fQmean2	no		$q_{\mathrm{mean}\,2}$	$1.18144E+00$	[pC]
fQwid	no		$q_{\mathrm{width}\,0}$	$1.42221E-01$	[pC]
fQwid1	no		$q_{\mathrm{width}\,1}$	$-1.65917E-01$	[pC]
fQwid2	no		$q_{\mathrm{width}\,2}$	$8.31019E-02$	[pC]
fEff	ϵ	0.99	ϵ_0	$9.98872E-01$	
fEff1	no		ϵ_1	$-4.68273E-02$	
fEff2	no		ϵ_2	$1.32732E+01$	
fEff3	no		ϵ_3	$7.04804E-01$	
fTime2Tdc	no		no		
fPedestal	no		no		
fQtoW0	no		no		
fQtoW1	no		no		
fQtoW2	no		no		
fQtoW3	no		no		
fGap	d	0.27	d	0.27	[mm]
fMode	0 or 1		0 or 1		

Table 1: Parameters needed by the digitiser for model I and model II. The names correspond to the Oracle database entries. The numerical values refer to Au+Au at 1.23 AGeV.

5 The digitiser

The digitiser is a task run by Hydra to produce the category holding the `HRpcCalSim` objects from the Geant output category holding the `HGeantRpc` objects. Beyond the obvious task of applying the detector response, the digitiser is responsible of changing the organisation of the information: there is one `HRpcCalSim` object for each track crossing a gap, which can hence appear multiple times, while there is only one `HRpcCalSim` for each active detector cell. This implies that multiple hits are also dealt with in the digitiser.

The digitiser is implemented in the `HRpcDigitizer` class. The parameters are hold in the class `HRpcDigiPar` that can be read and written from a `ROOT` file or initialised from Oracle with the standard facilities of the Hydra software for parameter container classes. A list of the actual parameters is given in Table 1.

The digitiser operates in four stages

1. The efficiency of each gap or of the cell is applied by generating a random number between 0 and 1. If the number is above the efficiency for that gap or cell, the gap or cell information is neglected and not further processed. The decision of employing the gap or cell efficiency depends if the `HGeantRpc` in use before or after June 2013 is found, respectively.

2. Depending on the value of the flag `fMode`, the digitiser can work gap wise (`fMode`= 0) or cell wise (`fMode`= 1), when the old version of `HGeantRpc` in use before June 2013 is found. In gap wise mode, the response is applied to each gap in the next stage and nothing is done in the present stage. In the cell wise mode, the gaps crossed by the same track are grouped together according to the physical cells before applying the response. This internal reorganisation of the information is accomplished by using essentially the same code present in `fillrpc`. Once all the content of the event has been reorganised, the gaps are combined by averaging the time of flight, the position along the module and the β of the particle. With the new version of `HGeantRpc` in use after June 2013, the cell wise mode is the only allowed one and the value of `fMode` is ignored. The mentioned reorganisation of the information is, of course, not necessary.

3. The detector response is applied to each gap or to each cell, depending on the output from the previous stage, producing hits. It should be appreciated that at this stage the hits are not yet the real detector hits as they may represent the contribution of a single gap (which is not observable) in gap wise operation or the contribution from one of several tracks. There are at present two models of the detector response, called model I and model II to be described afterwards.

4. The different hits from the previous stage are combined into a single final unique hit for each cell. This is again achieved by employing a C++ `<vector>` of structs of type `rpcdat`, addressed with a combined sector, column and cell number. The struct `rpcdat` internally holds a C++ `<vector>` of structs of type `gaptrack` for the left and an independent C++ `<vector>` for the right signals of the RPC cell. The times and charges are stored separately for the left and right sides. The different hits from the previous stage are combined into a final hit by taking the total charge and the smallest time (i.e. fastest signal) separately on the left and on the right sides. The only exception to this rule is the case of model II in gap wise operation, where the separate averages of the left and right times are used, as discussed in the section about model II. Some quantities like the array of Geant track numbers contributing to the left and right times are also calculated during this stage.

The main reason to operate the digitiser in a cell wise fashion is that only the final cell response is observable and hence can be empirically parametrised starting from real data. Because of the finite efficiency of each gap and because not all tracks cross all gaps, it is not trivial to attribute a response to each gap in such a way as to ensure that a predefined output is obtained for the final cell wise combination. Moreover, it is not clear that physically each gap acts as an independent detector, this is not certainly true at the level of the electronics response and noise. Still gap wise operation, especially in connection with the simple model I, can be useful to perform some basic studies, e.g. on the possible effect of the number of crossed gaps on the time resolution.

Two models are available for the response of the detector. Model I is the simplest description with constant efficiency, time and position resolutions and with a simple uniform charge distribution. It is useful mostly in situations when nothing is known. Model II includes refined parameterisations of the response with the energy loss of the particle, but requires experimental data to be tuned.

For backward compatibility, if the additional parameters necessary for model II are not initialised or not found in the Oracle database, they are set to 0 and the default model is I in gap wise or cell wise mode if the `HGeantRpc` in use before or after June 2013 is found, respectively. Model I in gap wise mode was the only available behaviour before December of 2012.

5.1 Model I

When necessary (i.e. `HGeantRpc` in use before June 2013 and `fMode`= 0), an inefficiency for each of the four gaps is derived from the full detector efficiency ϵ (see Table 1) according to

$$\bar{\epsilon}_{g\perp} = (1-\epsilon)^{1/4} \quad , \tag{1}$$

Both $1-\epsilon$ and $\bar{\epsilon}_{g\perp}$ are interpreted as referring to a particle crossing the gaps (and hence the cell) perpendicularly.

The processing steps applied by the digitiser to each hit in a gap or in a cell, depending on the flag `fMode`, are

1. This step may refer to an hit in a gap or in a cell, if the `HGeantRpc` in use before or after June 2013 is found, respectively, but for simplicity all quantities are indicated with a subscript g. The inefficiency of the gap is corrected to take into account the crossing angle of the particle

   $$\bar{\epsilon}_g = (\bar{\epsilon}_{g\perp})^{l_{\text{track}}/d} \quad , \tag{2}$$

   where l_{track} is the actual length of the track inside the gap as given by Geant and d is the gap thickness (see Table 1). Note that the energy loss in the gap is not taken into account. The efficiency per gap is calculated as

   $$\epsilon_g = 1 - \bar{\epsilon}_g \tag{3}$$

   and a random number r_1 uniformly distributed between 0 and 1 is drawn. If $r > \epsilon_g$ the hit is not processed further.

2. This step already happens after the second stage and can hence refer to an hit in a gap or in a cell, but for simplicity all quantities are indicated with a subscript g. The ideal left and right times referring to the left and right ends of the RPC cell are calculated from the geometrical dimensions and signal propagation speed v_{prop} (see Table 1) following

   $$\begin{cases} t_{g,l} = & t_G + x_{G,l}/v_{\text{prop}} \\ t_{g,r} = & t_G + x_{G,r}/v_{\text{prop}} \end{cases} \tag{4}$$

   where t_G, $x_{G,l}$ and $x_{G,r}$ are the time of flight given by Geant, the distance to the left end and the distance to the right end of the cell, respectively. The distance to the left end $x_{G,l}$ is actually calculated as the difference of the hit x coordinate given by Geant and the cell left corner x coordinate both in the sector reference system. The distance to the right end is $x_{G,r} = l - x_{G,l}$ where l is the cell length. The resolutions are added by drawing three random numbers: r_2 from a Gaussian distribution with a standard deviation σ_t (see Table 1) representing the intrinsic time resolution and r_3 and r_4 from a Gaussian distribution with a standard deviation σ_x/v_{prop} (see Table 1) representing the time resolution of the electronics. The real left and right times are then calculated as

   $$\begin{cases} t'_{g,l} = & t_{g,l} + r_2 + r_3 + t_{\text{off}} \\ t'_{g,r} = & t_{g,r} + r_2 + r_4 + t_{\text{off}} \end{cases} \quad , \tag{5}$$

   where t_{off} is a global time offset common to the whole RPC ToF wall. A limitation of the gap wise mode is that the electronics resolution is applied at the gap level. However, this behaviour has been kept for background compatibility with simulations made before December 2012.

3. The charge produced in the gap is sampled from a uniform distribution by drawing a random number r_5 uniformly distributed between 0 and 1 and calculating the quantity

   $$q_g = \frac{2\, r_5\, q_{\text{mean}}}{4\, l_{\text{track}}/d} \tag{6}$$

where q_{mean} is a unique mean charge for the full detector (see Table 1) and l_{track}/d, as in Eq.(2), is the correction for the track length inside the gap. Obviously, in this simple model, the charge is not related to the energy loss in the gap.

As mentioned, in the last stage, the final detector response is calculated by taking the earliest, independently for left and right sides, of all the processed gaps (i.e. containing an hit and passing the efficiency threshold) $t'_{g,l}$ and $t'_{g,r}$ and the sum of all the q_g.

5.2 Model II

Model II is an attempt to improve the description of data by actually parameterising the essential behaviours that have been identified so far. Its natural way of operation is, of course, cell wise but it can be used also gap wise for investigation purposes. In the latter case, corrections are provided to try to attribute the parametrised response of the cell to a gap. These are not exact because, during the processing of a certain gap, due to efficiency and geometry, it can not be known what is the fate of the other gaps composing the same cell. If it is desired that the final cell response strictly follows the input parametrisation, it is highly recommended to use the cell wise mode instead of trying to compensate by changing the value of the parameters in gap wise mode.

The choice has been made to parametrise the cell response as a function of β because, accordingly to Bethe-Bloch formula, this is the main dependence of the energy loss for charge one particles. Indeed the experimental data (see Figs. 2, 3, 4 and 5) show a substantial universality of the response once they are represented with this variable. Heavier fragments with the beam energies of HADES are very rare in the RPC wall acceptance and even more rarely reach the active volume of the detector. For completeness the digitiser treats them as all other charge one particles, but this case should rarely be of concern. The variable $\beta\gamma$ has been also tried to parametrise the response, but it expands the scale in a region where not much variation is present (i.e. in correspondence of the Fermi plateau or relativistic rise of the energy loss) while it compresses the scale in the region of low β where there is a strong variation.

Similar to model I, the processing steps applied by the digitiser to each hit in a gap or in a cell, depending on the version of `HGeantRpc` and on the flag `fMode`, are

1. The dependence of the efficiency from β is parametrised with a sigmoid function

   $$\epsilon = \epsilon_0 - \frac{\epsilon_1}{1 + e^{-\epsilon_2(\beta - \epsilon_3)}} \quad . \tag{7}$$

   Eq. (7) essentially has two saturation values ϵ_0 and $\epsilon_0 - \epsilon_1$ for $\beta = 0$ and $\beta = 1$ (actually reached only at infinity), respectively. The other two parameters describe the magnitude of the transition region $1/\epsilon_2$ and the position ϵ_3 of the transition between these two limiting values. All four parameters should be given as input (see Table 1).

   As in model I, when necessary (i.e. `HGeantRpc` in use before June 2013 and `fMode`$= 0$), an inefficiency per gap is derived from the full detector efficiency ϵ according to

   $$\bar{\epsilon}_{g\perp} = (1 - \epsilon)^{1/4} \quad , \tag{8}$$

   Again, both $1 - \epsilon$ and $\bar{\epsilon}_{g\perp}$ are interpreted as referring to a particle crossing the four gaps (and hence the cell) perpendicularly. The next step may refer to an hit in a gap or in a cell, if the `HGeantRpc` in use before or after June 2013 is found, respectively, but for simplicity all quantities are indicated with a subscript g. The inefficiency of the gap is corrected to take into account the crossing angle of the particle

   $$\bar{\epsilon}_g = (\bar{\epsilon}_{g\perp})^{l_{\text{track}}/d} \quad , \tag{9}$$

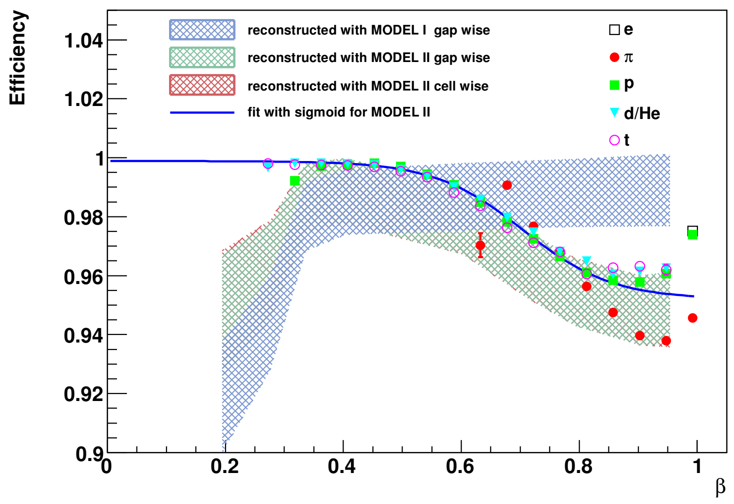

Figure 2: Efficiency of the RPC wall in Au+Au at 1.23 AGeV. The points are from a data analysis courtesy of G. Kornakov. The continuous line is the fit with a sigmoid function to the data points used to extract the parameters for Model II (see Table 1). The filled areas represent the reconstructed values from simulations with their statistical errors (red, green and blue shaded are refer to model I, model II in gap wise mode and model II in cell wise mode, respectively).

where l_{track} is the actual length of the track inside the gap as given by Geant and d is the gap thickness (see Table 1). The efficiency per gap is calculated as

$$\epsilon_g = 1 - \bar{\epsilon}_g \tag{10}$$

and a random number r_1 uniformly distributed between 0 and 1 is drawn. If $r > \epsilon_g$ the hit is not processed further.

2. This step already happens after the second stage and can hence refer to an hit in a gap or in a cell. As mentioned, for model II the natural choice is the cell wise mode. The left and right times are calculated similarly to Eq. (5) of model I according to

$$\begin{cases} t'_{c,l} = t_{c,l} + r_2 + r_3 + t_{\mathrm{off}} \\ t'_{c,r} = t_{c,r} + r_2 - r_3 + t_{\mathrm{off}} \end{cases} . \tag{11}$$

However now the random number r_2, representing the intrinsic time resolution, is drawn from a Gaussian distribution with standard deviation σ_t depending on β again according to a sigmoid function

$$\sigma_t = \sigma_{t\,0} - \frac{\sigma_{t\,1}}{1 + e^{-\sigma_{t\,2}(\beta - \sigma_{t\,3})}} . \tag{12}$$

All four parameters should be given as input (see Table 1). The second difference is that the random number r_3, again drawn from a Gaussian distribution with a standard deviation $\sigma_x / v_{\mathrm{prop}}$ (see Table 1), is now added on one side and subtracted from the other so that it does not influence the time resolution but it provides only the position resolution. The random number r_2 being added on both sides does not influence the position resolution but determines only the time resolution. In this way, the effects of the parameters $\sigma_{t\,0}$, $\sigma_{t\,1}$, $\sigma_{t\,2}$ and $\sigma_{t\,3}$ and of the parameter σ_x are totally uncorrelated and control directly the time and position resolutions

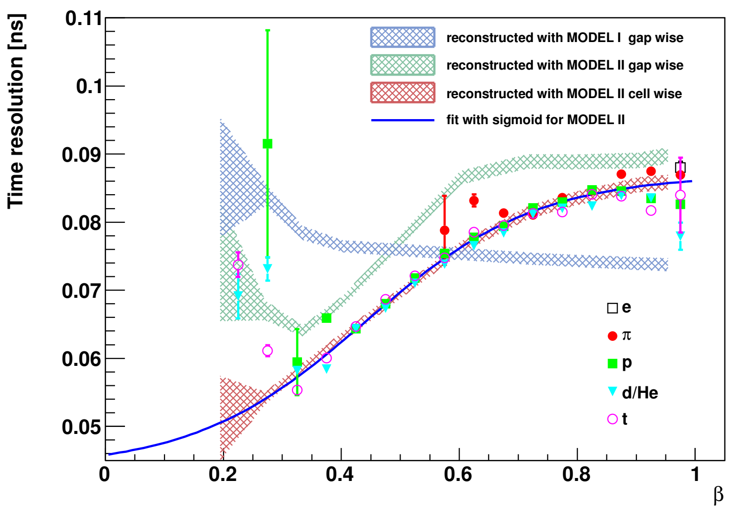

Figure 3: Same as Fig. 2 for the time resolution.

of the detector, respectively. If the gap wise operation is selected, a correction is applied by dividing both σ_t as given by Eq. (12) and σ_x by

$$c_t = \frac{\sum_{k=1}^{4} \binom{4}{k} \epsilon_g^k \,\overline{\epsilon}_g^{(4-k)} \frac{1}{\sqrt{k}}}{\sum_{k=1}^{4} \binom{4}{k} \epsilon_g^k \,\overline{\epsilon}_g^{(4-k)}} \quad , \tag{13}$$

to try to compensate on average for the number of gaps that will be contributing to the final response of the cell. This correction only represents an average and it can not be exact. The main difficulty is that in gap wise operation, when a given gap is being processed, the information about the other remaining gaps of the same detector cell is not available. Moreover, it does not take into account other factors influencing the number of active gaps like the curvature of the tracks due to the magnetic field.

3. The charge produced in the cell is sampled by drawing a random number r_5 according to a Landau distribution with most probable value q_{mean} and charge scale q_{width} (the name inconsistency in the parameters is due to constraint of maintaining the compatibility with the old convention meant for model I). It should be kept in mind that the Landau distribution does not have a well defined mean or r.m.s., but it has the advantage of representing well the experimental charge distributions and of being available amongst the predefined random number generators in `ROOT`. The dependence of q_{mean} and q_{width} from β is parametrised with two second order polynomials in β

$$\begin{cases} q_{\text{mean}} &= (q_{\text{mean}\,0} + q_{\text{mean}\,1}\,\beta + q_{\text{mean}\,2}\,\beta^2)\dfrac{l_{\text{track}}}{d} \\ q_{\text{width}} &= (q_{\text{width}\,0} + q_{\text{width}\,1}\,\beta + q_{\text{width}\,2}\,\beta^2)\dfrac{l_{\text{track}}}{d} \end{cases} . \tag{14}$$

The coefficients of the polynomials should again be given as input (see Table 1) and l_{track}/d, as in Eq. (9), is the correction for the track length inside the gap. If the gap wise operation is selected, again a correction is applied by dividing q_{mean} and q_{width} as given by Eqs. (14) by

$$\begin{cases} c_{q\,\text{mean}} &= 4\epsilon_g \\ c_{q\,\text{width}} &= \dfrac{\sum_{k=1}^{4} \binom{4}{k} \epsilon_g^k \,\overline{\epsilon}_g^{(4-k)} \sqrt{k}}{\sum_{k=1}^{4} \binom{4}{k} \epsilon_g^k \,\overline{\epsilon}_g^{(4-k)}} \end{cases} , \tag{15}$$

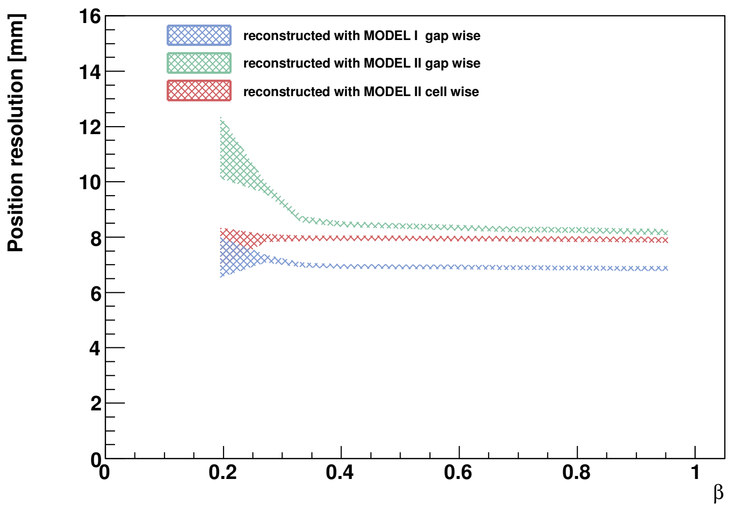

Figure 4: Reconstructed position resolutions from simulations with their statistical errors. The value employed for σ_x was 8 mm in all cases.

respectively, to try to compensate on average for the number of gaps that will be contributing to the final response of the cell. This is even more approximate than Eq. (13) because, beyond the already mentioned limitations, the properties of the self convolution of the Landau distribution with respect to its parameters are not so simple as for the Gaussian case.

As mentioned, in the last stage, the final detector response is calculated as in model I, except for the left and right times in gap wise operation where the average is taken. This is justified by two needs: the first is that the correction c_t should work and the second is to maintain the absence of correlation of the time and position contributions.

5.3 Validation of the digitiser

The present validation of the digitiser has been undertaken with a version of HGeant employing the the old `HGeantRpc` in use before June 2013. The figures shown here refer to such tests. It has later been checked again with the new version of `HGeantRpc` in use after June 2013 and, except for minor details, the cell wise mode, the only one available, reproduces again what is shown here.

To extract the value of the parameters for model II and validate the final result, the data from Au+Au at 1.23 AGeV were selected (courtesy of G. Kornakov). The comparison of efficiency, time resolution and position resolution are considered in Figs. 2, 3 and 4, respectively. The charge distributions for slices of β were both for data and simulations fitted with a Landau and the results for the most probable value and the charge scale are compared in Fig. 5. The values of the parameters for model II (see Table 1) have been extracted by fitting the experimental data for all $Z = 1$ particle species together (see the continuous blue line in Figs. 2, 3 and 5).

The simulations were performed with $4 \cdot 10^5$ events consisting of one proton per sector with a fixed momenta. The angular directions of the protons were changed at random to cover all the RPC ToF wall acceptance. The momenta were then varied for different simulation runs from 350 MeV/c to 3 GeV/c to scan with protons the same β range as covered in the data. The lowest point around $\beta \approx 0.2$ should not be taken too seriously since the protons barely punch trough into the RPC detector. In the experimental data, the value of β was derived from the time of flight and the path length and hence it depends on the full slowing down history of the particles, while in the

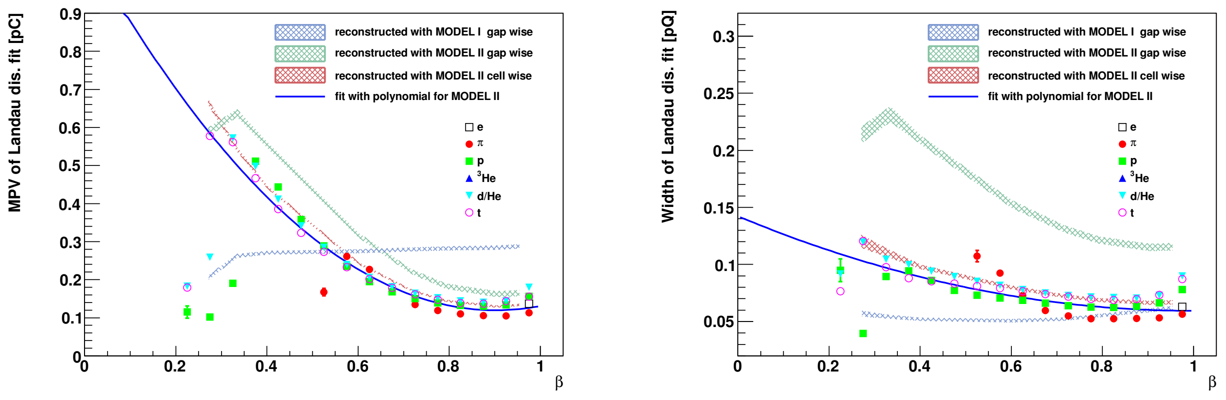

Figure 5: Parameters of the Landau fit in slices of β to the charge distributions in the RPC wall for Au+Au at 1.23 AGeV. The points are from a data analysis courtesy of G. Kornakov. The continuous lines are the fits to the data points with second degree polynomials used to extract the parameters for Model II (see Table 1). The filled areas represent the reconstructed values from simulations with their statistical errors (red, green and blue shaded are refer to model I, model II in gap wise mode and model II in cell wise mode, respectively).

simulations the digitiser uses the value of β at the entrance of the RPC gap. For all the higher points this difference is immaterial.

As expected, model I has a flat behaviour in all variables since it does not contain any explicit dependence on β. Model II in gap wise mode improves the situation and, finally, the agreement of model II operated in cell wise mode with the parameterisations used to fit the data (and hence with the data) is very good. The small differences visible in Fig. 5 between model II operated in cell wise mode and the parametrisation are due to the correction for the track length l_{track} inside the gap: because of the curvature given by the magnetic field l_{track} is on average slightly higher than the gap thickness d. The corrections c_t and c_q (see Eqs. (13) and (15), respectively) applied to model II in gap wise mode do indeed improve the situation, because without them, the disagreement with the parametrisation would be a factor of ≈ 1.5. In fact, even if the efficiency of the full detector is high, the average number of active gaps is around $2-3$. If all gaps were always active the correction factors should be around ≈ 2. The correction for the time and position resolutions is not perfect, as anticipated. The situation of the Landau distribution in the generated charge is worse (see green shaded area in Fig. 5), but this can be expected because of the mentioned difficulties with the properties of its self convolution. Of course the parameters could be changed to compensate for these effects, but this would be a tedious trial and error procedure. Model II in cell wise operations is always granted to reproduce the input parameterisations and should be preferred.

References

[1] Application Software Group, *GEANT 3: a detector description and simulation tool*, *CERN Program Library Long Writeup* **W5013** (1994).

[2] I.B. Smirnov, *Modelling of ionization produced by fast charged particles in gases*, *Nucl. Instr. and Meth. A*, **554** (1994) 474.