GSTAR: A Geant-based detector simulation chain for STAR

Peter Jacobs and Detlef Irmscher, LBNL

January 16, 1996

Abstract

This document describes a framework for Geant simulations of the STAR experiment, including the following packages:

- egz: Zebra-format output of event generators
- txgen: ASCII-format interface for user-generated events
- nml: definition of detector geometry and namelist-format interface to Geant
- gst: Core Geant program (GSTAR)
- g2t: assembly of GSTAR subevents and translation to Monte Carlo TAS tables

Contents

1 Introduction

This document describes the Geant-based detector simulation framework for STAR, called GSTAR, consisting of a chain from event input (both text-based and standard event generator-based) to Geant output in the form of DSL tables. GSTAR is based upon Geant version 3.21. This supplies several important advantages over previous versions of Geant: faster tracking, a PAW++ interactive interface, advanced graphics capabilities, and improved physics, especially in relevant areas such as delta electron production in gases.

GSTAR is derived to a considerable extent from GNA49, a similar package we wrote for the NA49 experiment. There is great similarity between the two experiments, and the essential "bookkeeping" and data-relation tasks are very similar. We have also undertaken some new developments for GSTAR (in particular in the area of passing detailed input event generator information through Geant to the output tables).

The geometry description utilizes the "envelope drawings" developed by the STAR engineers. It currently incorporates only the TPC (based upon STAR Note 203 plus later corrections) with a (currently constant) magnetic field. There are placeholders for all other subsystems, and for a magnetic field map. These will be implemented as they become available within the framework of GSTAR.

The output of the Geant simulation chain is now in the form of TAS tables. While ZEBRA is of course still used by Geant as its memory manager and therefore as the direct GSTAR output format, the user should immediately run an "afterburner" to GSTAR called g2t (Geant-to-TAS) which translates the Geant ZEBRA-format output to a set of TAS tables, described in section 5 of this document. The ZEBRA files should be temporary, existing only as long as it takes to translate them to tables; the TAS table output from g2t is what should be kept for long-term storage. This is for three important reasons:

- The output from GSTAR for a central Au-Au event is a few tens of MB. Given the data overhead in GEANT, it is impractical to perform the calculation of a complete event as a single GEANT "trigger", where all tracks from the event are held in memory at once. Nor is this necessary on physics grounds, since GEANT is a "single particle tracker". Consequently, the input event is broken up into subevents (typically consisting of 200 tracks each). These subdivisions have no physical relevance, and the subevents must be reassembled into a complete event to be usable. Keeping count of track and vertex numbering and hit associations during event reassembly is complex and fraught with pitfalls. It is highly desirable to have this done once, in a central place used by everyone. Since the direct GSTAR output must be altered anyway, it is most sensible to write it in the format in which it will be used downstream (i.e. TAS tables).

- GSTAR makes extensive use of GEANT "user buffers" for the tracks and vertices in order to pass information not passed by the standard Geant track and vertex access calls (gfkine, gfvert). These buffers are not self-describing and can be interpreted only if it is known a priori what they contain. On the other hand, tables are well-documented and stable.

- It is important to decouple from ZEBRA on general grounds. ZEBRA is an old system which will soon disappear. By decoupling the output of GSTAR from it now, the future change to a more modern memory management system will be transparent to the user.

The equivalent chain for NA49, GNA49 $\rightarrow$ g2ds, is in fact run as a named pipe during production of MC events. ZEBRA files are never written to disk, and exist only as virtual files for communicating along the Unix pipe.

A simple mechanism is implemented for merging two event streams of input to GSTAR, one consisting of events from a standard event generator (e.g. Hijing) and the other consisting of events made by hand (text format). Thus, the user can create by hand "events" consisting of a set of particularly interesting particles (ϕ's, D-mesons, close pion pairs, etc.) and embed them in a garden variety event to study efficiencies, resolutions, etc. in a realistic tracking environment.

The output tables are designed to support more general merging tasks, utilizing the concept of a "superevent". An important application of the superevent is the creation of events including the out-of-time minimum bias background, to simulate event pileup at high luminosity (section 5.4).

GSTAR has the capability of passing detailed event generator information to the output tables, including the cascade of parents interacting or decaying to daughters. We have built in a prototype mechanism (which still requires some development) to begin discussion and work in this area. Its use is best illustrated by examples:

- Event generators such as RQMD, where a significant fraction of the meson production goes through resonance matter. Knowledge of the kinematics of the parent resonances and their daughters is important to study the dynamics of this resonance gas and how it is reflected (or not!) in the measured pion spectra of final state particles,
- Jet production from a string system, where it would be useful to tag all final state products of the hard partonic scattering to distinguish them from products of other, incoherent soft production processes,
- the study of rare or high multiplicity decay modes (GEANT cannot handle greater than 3-body decays).

We do not mean to imply that we are able at present to extract full kinematical and parentage information from RQMD, Hijing, etc., but we want to begin a discussion of how to do this and whether the current mechanism has enough flexibility to do the job. This is probably not the case: currently we cannot handle vertices having more than one parent (i.e. 2-body or n-body interactions, string systems with many labels, etc.), though this capability has been designed into the output tables. This is work in progress.

In addition to the Zebra and ASCII formatted event input, some simple generators have been implemented internally in GSTAR to allow the user to generate events without having to supply an input file. These currently consist of a uniform distribution within either a $p_{\perp}$-y phasespace bin or a three-momentum bin (see sections 2, 8.3).

GSTAR contains a facility ("udecay") to define the decay of a given particle type into a single decay mode, with decay vertices distributed uniformly between two spatial boundaries instead of exponentially from the point of creation of the parent. This increases the "efficiency" of the Monte Carlo for studying, for instance, the reconstruction efficiency of weakly decaying particles, where only a small fraction of the decay vertices of physically generated particles may fall within the fiducial volume (and therefore most of the generated tracks are "lost" to the efficiency study).

GSTAR contains a facility to define a new particle and its decay modes within a kumac ("spartstar"). This is in principle supplied as a standard GEANT command ("spart"), but the CERN Library version contains bugs.

From the end-user's standpoint, the most important issue is the definition of the TAS table contents. In section 5 we present the tables constructed to sort and transmit the wealth of information available from the Monte Carlo in a fashion that is convenient for a variety of types of analysis.

This is accomplished by a set of somewhat redundant bidirectional pointers (using primary keys) amongst tracks, vertices, and detector hits in the various subsystems. The guiding principle in designing the tables was to be highly redundant for convenience where this is not expensive (vertices and tracks), and concise where redundancy is expensive (hits).

For a central Au-Au event with all physics processes turned on (but tracking of δ-electrons only above 1 MeV) and TPC hits only recorded, the output file size is 27 MB, of which 17 MB are TPC hits, 8 MB are tracks, and 2 MB are vertices. This event took about 5000 cpu seconds to process on rsgi00 (compiled for debugging and hence not optimized for time). Turning off hadronic physics processes reduces this time to about 1200 seconds.

We now turn to a description of the components of the GSTAR chain (see Fig. 1). We will then give instructions how to run the various codes, and indicate some useful Kumacs. All code referred to currently resides in the publicly accessible area

`/afs/rhic/star/starlib/dev/sim/geant .`

For convenience we will refer to this directory as **$GEANT_LIB** below. This environment variable is set in the file setup_gstar (see section 8).

2 Event Input

Event input to GSTAR is either from an external event file or from one of a set of simple internal generators that have been implemented within GSTAR.

External files of events can be (i) generated by event generators (Hijing, Venus, RQMD, etc.) and written in a standard way (currently utilizing the egz package), or (ii) hand-made by the user and stored in a standard ASCII text format (utilizing the txgen package). Input to GSTAR from external files can come from an event generator file, a text file, or from both, with the two events merged into one. During the merge care is taken to ensure an unambiguous labeling for tracks and vertices to avoid overlaps between the events. The output tables contain sufficient information to allow the merged events to be easily disentangled downstream of Geant.

A set of simple internal generators has been implemented in GSTAR. These are invoked by a GSTAR command line (see section 8). Currently there are two internal generators available:

- Momentumbin: throws a given number of tracks having a single Geant particle id uniformly within a defined 3-momentum bin, originating from the first primary vertex,
- Phasespace: throws a given number of tracks having a single Geant particle id uniformly within a defined $p_\perp$-y bin, originating from the first primary vertex.

Additional simple generators can be implemented if there is a demand for them (e.g. exponential $p_\perp$ spectrum, mix of Geant PIDs, etc.).

Standard event generator input is created via the existing package egz with recent modifications (see section 7). The file format remains ZEBRA for the time being, though the event input routine to GSTAR currently puts the events into structures which can be considered to be prototypes of future DSL tables containing event generator data. However, egz does not currently have the capability to record the detailed dynamical event generator information, and we do not intend to modify it to do so. Full implementation of this capability for event generators must await final definition of these tables and the development of a new mechanism to fill them. A more detailed discussion of the event input (Sections 6 and 7) follows the discussion of the GSTAR output table contents.

3 Geometry Input

The namelist file mechanism remains, for the present, the means by which geometry is communicated to GSTAR. All namelist files should be generated using the package nml (slightly modified from the mechanism described in STAR Note 85). This mechanism allows all coordinates and dimensions to be calculated from a database, making numerical changes trivial, as well as helping to ensure consistency between MC event generation and tracking geometry parameters. In addition, it makes the namelist files portable among operating systems having difference conventions for namelists (the Fortran codes should simply be recompiled and run on the system to be used for GSTAR). A diagnostic Kumac is generated in parallel with the namelist file, for use in visual inspection of the geometry while running GSTAR (see section 3.1).

The namelist files to be used are specified to GSTAR by defining a set of Unix environment variables. Different namelist files for a subsystem can be used simply by changing the environment variable before running

GSTAR, or the subsystem can be deleted entirely by defining the appropriate environment variable to be NULL.nml.

Currently there are two non-NULL namelist files defined: CAVE and TPC.

The STAR detector is described in hierarchical fashion, with each subsystem placed within an "envelope", rather than directly into the CAVE volume, which contains all subsystem envelopes. The set of envelopes is currently based upon the STAR Engineering envelope drawing supplied by Doug Fritz of LBNL (version dated 6/15/94), which details the boundaries enclosing each subsystem. GSTAR should follow the engineering conventions and dimensions as closely as possible, but this is not entirely straightforward. There are some boundaries that have to be negotiated, such as whether the inner insulating gas of the TPC belongs to the SVT volume. Whatever is decided on the engineering level should be reflected in GSTAR. Numbering and labelling conventions follow STAR Note 229.

The four-character name of each volume of the subsystem should begin with a letter unique to that subsystem (the exception is "CAVE", which must be defined, whose first letter overlaps with that of the EMC barrel). This assignment is of course arbitrary:

```
T: TPC
S: SVT
P: Beam Pipe
R: TPC Readout Electronics
B: Central Trigger Barrel and TOF
C: EM Calorimeter Barrel
E: EM Calorimeter Endcap
M: Magnet Pole Tip
L: EMC eLectronics
V: VPD
X: XTPC
F: SVT Support Cone ("Facilities for SVT")
```

The set of subsystem envelopes placed in the CAVE volume is shown in Fig. 2, produced by the GEANT command:

```
GEANT> dtree CAVE 2
```

Those volumes with "*2" in their boxes have two copies placed in the CAVE.

Fig. 3 is a cut away view of the envelopes, produced by the Kumac $GEANT_LIB/gstar/exa/draw_star_low_resolution (see section 9).

TPC: This namelist file implements the description of the TPC based upon STAR Note 203, with later modifications for volume definitions, sector orientation, and numbering. All parameters are currently defined in an include file specific to the namelist generation. It is intended to extract them instead from a general (to be defined) geometry database, to ensure consistency with tracking code. Detailed structure of the endcaps, including readout electronics and aluminum supports, is implemented. The smallest readout volume is the width of the physical padrow, not of the sense wire volume (as was used previously for the slow simulator).

The complete tree of volumes rooted in TPC is shown in Fig. 4. For association with physical structures, see STAR Note 203. Figures 5 - 7 show various components of the TPC and its endcap, seen end-on. Fig. 5 shows the padrow volumes and field cages. Fig. 6 shows the aluminum sector structure, while Fig. 7 shows the aluminum support wheel.

3.1 Namelist file generation (package nml)

Code for the shell, a Makefile, and example user code that builds the CAVE and TPC namelists are found in $GEANT_LIB/gstar_geo.

Each volume has a drawing colour assigned to it (using the GSATT argument 'COLO') that is a function of the density of its assigned tracking medium. This is useful for finding errors in the geometry descriptions. For details see the subroutine **nml_set_attributes** in **nml_shell.F**, but generally speaking the following colour assignments are made:

black outline: air, vacuum

red: metals such as aluminum, beryllium, iron, copper

green: plastics, glue, g10, etc.

violet: dense nonmetallic substances (silicon, carbon, etc.)

blue: sensitive volumes filled with gas (p10, He-ethane, etc.)

light blue: nonsensitive volumes filled with gases other than air

These assignments can of course be overridden in a drawing Kumac.

A general problem experienced by anyone who works with Geant is the absence of debugging tools for geometry. A new functionality has been built into the namelist generation shell to generate a diagnostic Kumac in parallel to the actual namelist file. The Kumac sets all volumes invisible, and then one by one, under control of the user, makes the volumes of the subsystem visible and draws them (see section 9). The user would first draw a subsystem view at a desired angle and magnification, and then run the Kumac to examine the relative placements of all the defined volumes of the subsystem. Note that only those volumes for which **nml_set_attributes** has been called will have a corresponding entry in the Kumac (for example, see nml_create_tpc.F).

3.2 GSTAR Tracking Media

In order to describe continuous processes correctly (principally multiple coulomb scattering and energy loss), the GEANT step size within a thin volume should be somewhat less than the width of the volume traversed during the stepping. As a first attempt to try to ensure that this is done correctly, tracking media defined within GSTAR are assigned to one of three classes: gases, thin dense media, and thick dense media. The thin dense media are those which a significant fraction of tracks might cross before entering a sensitive detector. Currently the maximum step size in thin media is set to 20 μm. Geant defaults are taken for most of the remaining medium parameters. To get a complete list of GSTAR-defined tracking media and their parameters, run GSTAR and type:

```
GEANT> print tmed 0
```

4 GSTAR: Core Geant Code

4.1 Event Input

The event generator file may contain a complex history describing the development of the event on the fermi length scale. GSTAR must know which among the tracks in the file to track and which to simply pass to the output tables untouched. We introduce here the terminology of a "final state" track from the generator, being one that is handed off to GSTAR to be tracked. The current default is that all tracks with (i) a pid defined with GSTAR (and therefore recognizable to it), and (ii) having a lifetime at least as long as the π°, are defined as final state tracks. We foresee a simple mechanism (not yet implemented) to allow the user to override the default and have GSTAR track and decay shorter-lived particles (e.g. ϕ's), given that they are defined with decay modes within GSTAR. This solves the age-old problem of knowing what the event generator has decayed and what Geant should decay while avoiding double counting: let the generator decay everything, and GSTAR will select what it recognizes and should track. In the future we will be investing large amounts of CPU time into the generation of events. This mechanism will allow multiple uses of a given generator event, enhancing the efficiency of MC studies.

4.2 Shower Tracks

Special treatment is given to what we call "shower tracks". In the case of the development of a shower in uninstrumented dense material (e.g. calorimeter absorber, magnet iron), it is undesirable to write out all the intermediate tracks in the shower that do not escape the dense material. However, a track from the shower may escape the material and generate hits in a sensitive detector. These background hits must of course be recorded in the sensitive detectors, but in the standard GEANT stacking mechanism the track that generated them is not recorded. We save these tracks in GSTAR in the user buffer of the track responsible for generating the shower, extracting them in g2t and associating them with the hits for which they are responsible. Some information is necessarily lost in the process, for instance where precisely the shower track was created and by what mechanism, but this information is relatively unimportant. Most important is the connection of the hits to the shower track and the knowledge of the track's id, momentum, and shower-initiating parent, which are available in the output tables.

4.3 TPC Hits

A track crossing a padrow in the TPC most often traverses a full pad length while moving radially outward or inward. In such a case, GSTAR calculates the transverse distance between entrance and exit points on the padrow (i.e. length of difference between entrance and exit vectors, projected perpendicular to the radial vector). If the transverse distance is less than a few centimeters (this cut is currently 3 cm.), one hit is generated for that track on that padrow, with radial coordinate at the center of the padrow volume and transverse coordinates determined by a quadratic interpolation between entrance and exit points. The interpolation utilizes the points and momenta at both the entrance and exit. Energy loss is integrated over the total path length of the track in the padrow. In the rarer case of the track traveling transversely a considerable distance within the padrow (i.e. greater than the transverse distance cut), several hits are generated along the trajectory, not necessarily at the radial center of the padrow. Energy loss for each hit is that for the track fragment the hit represents. There is special treatment for a delta electron in the TPC gas: since delta electrons typically suffer very large effects from multiple coulomb scattering in gases, the above transverse distance cut is reduced to 3 mm.

These mechanisms are illustrated in Fig. 8 which shows an x-y view of all hits from a central Au-Au event on several padrows near the middle of sector 12. All hits are shown with equal weight (i.e. energy loss is not represented in the figure). The large majority of hits are due to tracks which enter a padrow from below and exit it above; these are placed at the center of the padrow. Also seen are hits due to tracks which enter the padrow via the lower or upper surface and exit through the left or right edges or into the endcap or midplane; these are scattered throughout the padrow, with position weighted by the actual trajectory of the track. Finally, delta electron tracks are seen, with hits spaced by 3 mm transverse displacement along their trajectories.

5 g2t: Translation of GSTAR Output to TAS Tables

Given the above definitions and concepts, we turn now to the new proposal for output tables from GSTAR $\rightarrow$ g2t. We construct a bidirectional web of pointers (in the world of TAS tables, primary keys) connecting tracks to their parent and daughter vertices and their hits, vertices to their parent and daughter tracks, and hits to their tracks. Additionally, information must be supplied specifying whether a given track or vertex was created by the event generator or GSTAR.

We use linked lists to connect objects having a common property (e.g. hits belonging to one track, all tracks that are daughters of the same vertex, etc). Pointers to objects refer to their primary keys. Thus, if a set of tables is edited, the connections remain correct, even though the array indices are not. The end of a linked list is indicated by a pointer to id=0 (equivalent to a NULL pointer).

Note that there is no significance given to the place of an object in a linked list, even for those objects for which an obvious, prefered ordering might exist. For instance, in the linked list of hits associated with a track in the TPC, the hits are not necessarily ordered from first to last in time, or from smallest to largest radial coordinate. It is the user's responsibility to order them as needed.

For the sake of simplicity, the following tables are presented here in the form of C-structures. This is appropriate, since these closely match the table definitions required by DSL. They can easily be mapped into Fortran structures by automated procedures.

Note that not all table entries are currently being filled by GSTAR/G2T, in particular those passing information from the event generators. However, we have written the mechanisms and reserved appropriate entries in the tables in anticipation of the information from the event generators becoming available.

5.1 Run Table

This is information that is constant over the run and need not be repeated every event.

```
typedef struct
{
long            generator;      /* Event generator id */
char            eg_name[32];    /* Event generator name */
float           eg_version;     /* Event generator version */
long            eg_run;         /* Event generator run number */
long            eg_rndm[2];     /* Event generator initial random seeds */
float           sqrts;          /* nucleon-nucleon center of mass energy */
long            is_minbias;     /* flag if minimum bias run */
float           b_min;          /* Min. impact parameter */
float           b_max;          /* Max. impact parameter */
long            east_a;         /* east beam mass number */
long            east_z;         /* east beam proton number */
long            west_a;         /* west beam mass number */
long            west_z;         /* west beam proton number */
float           geant_version;  /* GEANT version number */
long            ge_run;         /* GEANT run number */
long            ge_rndm[2];     /* GEANT initial random seeds */
long            events_ordered; /* requested number of events (>= actual) */
long            date;           /* Creation date */
long            time;           /* Creation time */
char            author[80];     /* username of author */
char            machine[80];    /* contains uname -a */
```

```
long            event_mixing[5];/* info on event mixing - to be specified */
float           polarization[10];/* polarization state  - to be specified */
long            reserved[5];    /* Reserved for future applications */
} G2t_run_row_st;
```

Event **generator** id is an integer uniquely identifying each generator. Current conventions are (in accordance with egz):

```
TEXT or internal    0
HIJET               1
VENUS               2
FRITIOF             3
HIJING              4
FRITIOF17           5
```

Further generators (e.g. RQMD) will be added in the future.

Definition of entries pertaining to event mixing and beam polarization await input from the relevant communities.

5.2 Particle Table

This specifies the particles defined within GSTAR for this run. Standard Geant pids are given in the Geant v3.21 manual, CONS300-1. They will be contained in the particle table, which will also contain non-standard particles defined specifically for GSTAR. The sequence of pids does not form a dense set, so that the pid value corresponds to the table entry number. This is a slight waste of space, but makes addressing easier. Unused entries are set to zero. This table is filled once at the beginning of the run. There is one word reserved for future use.

At the moment, pdg_pid and eg_pid are not filled.

For non-stable particles, the number of decay modes and a pointer to the first in a linked list of decay modes are given; these are derived from the GEANT decay modes (see CONS310-1 in the GEANT3.21 manual).

```
typedef struct
{
long            pid;            /* GEANT particle id */
char            napart[32];     /* Name of particle */
float           amass;          /* Mass (GeV/c^2) */
float           charge;         /* Charge in e units */
float           tlife;          /* Lifetime in sec */
```

```
long            pdg_pid;        /* PDG particle id */
long            eg_pid;         /* event generator particle id */
long            n_modes;        /* number of decay modes */
long            mode_p;         /* Id of first decay mode */
long            reserved;       /* for future use */
} G2t_gepart_row_st;
```

Symbolic names are assigned to all defined particles in the include files "$GEANT_LIB/gstar/inc/gstar_pid.inc" and "gstar_pid.h". We strongly recommend that these files be included explicitly in any text-format generator, rather than hard-wiring the Geant pids into the code, thus facilitating future changes and additions.

All particles that were defined in mcg are included, with the addition of the "laserino", a particle having the geantino tracking type (i.e. chargeless, totally non-interacting) but which nevertheless will leave hits in the TPC gas.

5.3 Decay Tables

All decays known to GSTAR are filled in this table, which forms a linked list of decay modes for a given particle type. The daughters for a given mode are kept in a linked list. There is therefore no restriction on the number of decay modes or number of daughters per mode that can be described. In future these tables will also be used for describing decay modes defined within the event generators.

```
typedef struct
{
long          id;                /* primary key */
float         b_ratio;           /* branching ratio for this mode */
char          mode[80];          /* character string describing mode */
long          mode_label;        /* integer labelling mode */
long          n_daughter;        /* number of daughters of decay */
long          daughter_p;        /* Id of first daughter */
long          next_decay_p;      /* pointer to next decay mode */
} G2t_decay_mode_st;
```

```
typedef struct
{
long          id;                /* primary key */
long          ge_pid;            /* geant pid */
```

```
long          eg_pid;           /* event generator pid */
long          next_daughter_p;  /* next daughter of same decay mode */
} G2t_decay_daughter_st;
```

5.4 Superevent Table

This table is used to associate several GSTAR/G2T output events together, for instance to represent a physical event in the STAR detector at high luminosity which includes non-trigger minimum bias pileup events. This merging of events must be done downstream of GSTAR/G2T, where the physical hits are offset in space by the appropriate drift velocity times the time difference between the background event and the trigger. However, by supplying every GSTAR/G2T output event with a superevent table (pointing initially to a single event, and hence not strictly necessary), code using these events as input does not have to be sensitive to whether the merging has been performed or not.

```
typedef struct
{
long            n_events;       /* number of events in superevent */
float           earliest_event; /* time of earliest ev. wrt trigger ev. (<0) */
float           latest_event;   /* time of latest ev. wrt trigger ev. (>0)   */
} G2t_superevent_row_st;
```

5.5 Event Table

This contains information that changes from event to event. Additionally it contains a pointer to the start of the linked list of primary vertices (a primary vertex is a vertex that does not have a parent track). Polarization and merge_info are provided as placeholders for later additions.

```
typedef struct
{
long            n_run;          /* Run number */
long            n_event;        /* Event number */
long            ge_rndm[2];     /* geant random seeds at start of event */
long            eg_label;       /* Event label from event generator */
float           b_impact;       /* Impact parameter */
float           phi_impact;     /* Azimuth of reaction plane */
long            event_type;     /* specifies trigger, pileup, cosmic, laser */
float           time_offset;    /* time offset in seconds wrt trigger event */
```

```
long            polarization[10]; /* polarization info, as yet unspecified */
long            merge_info[10]; /* as yet unspecified */
long            n_part_prot_east; /* number of participant protons */
long            n_part_neut_east; /* number of participant neutrons */
long            n_part_prot_west; /* number of participant protons */
long            n_part_neut_west; /* number of participant neutrons */
long            n_track_prim;   /* Number of primary tracks (i.e. w/o parent)*/
long            n_track_eg_fs;  /* Number of final state event gen. tracks */
long            prim_vertex_p;  /* Id of first primary vertex in event */
} G2t_event_row_st;
```

Values for entry **event_type** have not yet been specified.

5.6 Vertex Table

The vertex table contains information about the vertex and its connection to its parent and daughter tracks.

parent track(s): In Geant, every vertex has only one parent track (or none for a primary vertex). However, in event generators, a vertex can have two or more parents. We have built in the capability of describing such vertices by specifying **n_parent**, the number of parents of the vertex. **parent_p** points to the start of the linked list of parent tracks, which is followed via **next_parent_p** in the track structure. In the case of a Geant-created vertex, **n_parent** will be 1 and **parent_p** will point to the sole parent track, whose **next_parent_p** will be 0. At present the interface to the event generators is not capable of extracting multiple parentage of vertices - this is a project for the future.

The nature of the vertex can be determined from **ge_proc** and **eg_proc**, only one of which will be non-zero. If the vertex was created by the event generator, then **ge_proc** will be zero and **eg_proc** will contain the generator-specific integer code corresponding to the physical process responsible for creating the vertex. If the vertex was created by Geant, then **ge_proc** will be nonzero and contain the Geant mechanism code, specified in the Geant v3.21 manual, pg ZZZZ010-22 (description of common block GCTRAK). A few of the most common Geant codes are 5 for decay, 6 for pair production, 10 for delta electron production, and 12 for hadronic interaction.

eg_label is a generator-specific unique integer label assigned to a vertex created by the event generator. Geant does not use this quantity. It is passed downstream for possible use in reconstructing the event generator event or at least checking its consistency, or as a user-defined vertex flag.

We introduce the concept of an **intermediate vertex**. An intermediate vertex of a track is one created by the track via a process that does not result in the killing of the track: Bremsstrahlung, delta electron production, inelastic scattering, etc. An intermediate vertex is flagged by **is_itrmd**.

There are two times of flight specified. **eg_tof** is the production time of the vertex in the event generator, if it is a generator vertex, and is likely to be on the physical scale of fm/c (10^{-23} sec). This can be used together with **eg_x[3]** to record space-time coordinates of particle production, useful for instance in HBT studies. **ge_tof** is the production time of the vertex if it was created by Geant, relative to the earliest vertex in the event. **ge_tof** is likely to be on the scale of nanoseconds. For a generator-created vertex, **ge_tof** and **ge_x[3]** are zero; for a Geant-created vertex, **eg_tof** and **eg_x[3]** are zero.

daughter_p points to the beginning of a list of **n_daughter** entries in the g2t_track table, which can be followed via **next_vtx_trk_p**. The pointer **next_itrmd_p** navigates the list of intermediate vertices created by the parent track. **next_prim_v_p** navigates the linked list of primary vertices in the event (i.e. those with no parent track), the first of which is pointed to by the event table.

The remaining quantities are self-explanatory.

```
typedef struct
{
long            id;             /* primary key */
long            event_p;        /* pointer to event */
long            n_parent;       /* number of parent tracks */
long            parent_p;       /* Id of first parent track */
long            ge_proc;        /* >0 GEANT, =0 event generator */
long            is_itrmd;       /* flags intermediate vertex */
float           ge_tof;         /* GEANT vertex production time */
float           ge_x[3];        /* GEANT vertex coordinate (Cartesian) */
long            ge_medium;      /* GEANT Medium */
char            ge_volume[4];   /* GEANT volume name */
long            eg_label;       /* generator label (or 0 if GEANT vertex) */
long            eg_proc;        /* generator production mechanism (if any) */
float           eg_tof;         /* generator vertex production time */
float           eg_x[3];        /* generator vertex coordinate (Cartesian) */
long            n_daughter;     /* Number of daughter tracks */
long            daughter_p;     /* Id of first daughter in linked list */
long            next_itrmd_p;   /* Id of next intermediate vertex */
long            next_prim_v_p;  /* Id of next primary vertex */
} G2t_vertex_row_st;
```

5.7 Track Table

There are two particle ids supplied for each track, at least one of which is non-zero. **ge_pid** is non-zero for particles known to GSTAR, as given in the particle table described in section 5.2. If a generator track is not recognized by GSTAR, **ge_pid** will be zero and the particle will not be tracked by GSTAR, but simply passed to the output table. **eg_pid** is a generator-specific integer label for tracks created by the event generator, and is simply passed to output by GSTAR.

eg_label is a unique integer label assigned to generator-created tracks by the generator. It is zero for Geant-created tracks, and can be used as a flag to distinguish between these two classes of tracks.

The kinematic quantities are self-explanatory. For non-primary vertex tracks, **rapidity, eta** and **pt** are set to -999.

is_shower is a flag indicating that a track is a shower track (=1) or not (=0). For definition of shower tracks see section 4.2.

A track necessarily has a vertex of origin, pointed to by **start_vertex_p**.

If a track is killed within the detector, the vertex where this occurs is pointed to by **stop_vertex_p**. If the track exits the setup without being killed, stop_vertex_p=0.

In the case of multiple parents of a vertex (not yet implemented), **next_parent_p** navigates the linked list of all parents of their common stop vertex.

In the frequent case that a vertex has several daughters, **next_vtx_trk_p** navigates the linked list of all daughters having the same start vertex. Note that this is used both for the simple case of a 2-body decay vertex and the case of the primary event vertex producing 20K tracks.

In the case that this track generates intermediate vertices (for definition see 5.6), **itrmd_vertex_p** points to the beginning of the linked list of these vertices.

We supply here hit counters and pointers to the start of linked list of hits originating from the track for eight subsystems (tpc, svt, xtp, emc, vpd, tof, ctb, msc), though currently only tpc hits are filled. The remaining tables have not yet been finalized.

```
typedef struct
{
long            id;             /* primary key */
long            ge_pid;         /* GEANT particle id */
long            eg_label;       /* generator track label (0 if GEANT track)*/
```

```
long            eg_pid;         /* event generator particle id */
float           charge;         /* Charge */
float           p[3];           /* Momentum */
float           e;              /* Energy */
float           rapidity;       /* Rapidity */
float           eta;            /* Pseudorapidity */
float           pt;             /* Transverse momentum */
float           ptot;           /* Total momentum */
long            is_shower;      /* 1 if shower track, 0 if not */
long            start_vertex_p; /* Id of start vertex of track */
long            stop_vertex_p;  /* Id of stop vertex of this track */
long            next_parent_p;  /* Id of next parent track */
long            next_vtx_trk_p; /* Next daughter track of start vertex */
long            itrmd_vertex_p; /* First intermediate vertex */
long            n_tpc_hit;      /* Nhits in tpc */
long            n_svt_hit;      /* Nhits in svt */
long            n_xtp_hit;      /* Nhits in xtpc */
long            n_emc_hit;      /* Nhits in emc */
long            n_vpd_hit;      /* Nhits in vpd */
long            n_tof_hit;      /* Nhits in tof */
long            n_ctb_hit;      /* Nhits in ctb */
long            n_msc_hit;      /* Nhits in msc */
long            hit_tpc_p;      /* Id of first tpc hit on track linked list */
long            hit_svt_p;      /* Id of first svt hit on track linked list */
long            hit_xtp_p;      /* Id of first xtp hit on track linked list */
long            hit_emc_p;      /* Id of first emc hit on track linked list */
long            hit_vpd_p;      /* Id of first vpd hit on track linked list */
long            hit_tof_p;      /* Id of first tof hit on track linked list */
long            hit_ctb_p;      /* Id of first ctb hit on track linked list */
long            hit_msc_p;      /* Id of first msc hit on track linked list */
} G2t_track_row_st;
```

5.8 TPC Hit Table

These are by far the most expensive tables for storage, and should be kept compact.

parent_p is the label (or primary key) of the parent track.

volume_id marks the volume the hit is located in, using $1000 \cdot n\_sector + n\_padrow$, where n_sector and n_padrow are follow the conventions of STAR Note #229.

Coordinate and local momentum are self-explanatory. **x[3]** is given in the global STAR coordinate system.

For **de**, see discussion of tracking in the TPC in section 4.3. If the track traverses the pad row (most common case), the hit is placed its center and the energy deposition is positive. Otherwise, the negative of the value of the energy deposition is given.

The distance a particle traveled within a padrow to generate this hit is **ds**. The energy loss integrated over this trajectory is **de**.

tof can be used for ordering hits on a track. Note that successive hits in the linked list are not necessarily ordered in tof.

next_tr_hit_p navigates the linked list of hits associated with the same track.

```
typedef struct
{
long            id;             /* primary key */
long            track_p;        /* Id of parent track */
long            volume_id;      /* STAR volume identification */
float           x[3];           /* coordinate (Cartesian) */
float           p[3];           /* local momentum */
float           de;             /* energy deposition at hit */
float           ds;             /* path length within padrow */
float           tof;            /* time of flight */
long            next_tr_hit_p;  /* Id of next hit on same track */
} G2t_tpc_hit_row_st;
```

N.B.: We reiterate that the hits are **not** ordered in any manner. Any desired ordering has to be done by the subsequent user!!

6 Text Input Format

This section describes an ASCII text input format designed to fulfill two tasks:

- to be a simple, quick means to put a hand-made set of tracks into GSTAR, and
- to be a mechanism for transmitting the full range of information that can be represented in the above tables, and therefore be a kind of prototype general interface to event generator information of arbitrary complexity.

6.1 Line Formats

There are four kinds of lines in the text file, plus arbitrary interspersed comments.

- Kinematic information about the run of events may be passed to the output tables via the following line:

  GENER: eg_name eg_nr eg_vers z_proj a_proj z_targ a_targ sqrt_s bmax frame

  If the GENER: line is not present, ASCII text format using Geant particle codes with kinematics calculated for a collider is assumed, so that for text files this line is optional. (It will be used for event generators equipped with the option to output data in this format to specify the translation scheme from event generator particle codes to those of Geant.)

  - **eg_name** should be 'TEXT', **eg_nr** should be 0.

  - **z_proj** etc. are the charge and mass of "projectile" and "target"

  - **sqrt_s** is CM energy per nucleon pair

  - **bmax** is the maximum impact parameter used to generate the events

  - **frame** is either "CMS" or "LAB"

- Each event must begin with the line

  EVENT: event_id n_tracks n_vertices

  format: ('EVENT:', 3x, 3(1x,i6))

  where **event_id** is an integer giving a label to the event which is unique within the file, **n_tracks** gives the number of tracks in the file associated with the event, and **n_vertices** gives the number of vertices. It is not necessary to specify any vertices, so that **n_vertices** = **0** is allowed. In this case all tracks are understood to be emitted from the same vertex positioned at the origin.

  The **event_id** should be a positive integer. The file should be terminated with an event line having **event_id** = **-999.**

- Each track is specified by a line

  TRACK: ge_pid px py pz [track_id start_vertex stop_vertex eg_pid]

format: ('TRACK:', 1x,i6, 3(1x,g12.5), 4(1x,i6))

where the four fields in brackets are optional, though if one is present they all must be present.

- **ge_pid** is the Geant PID code for the track, or zero if its particle type is not known to GSTAR (in which case it will be passed through Geant to the output tables without being propagated through the detector)

- **px, py, pz** constitute the three-momentum of the track

- **track_id** is an integer label of the track which is unique within the event. It is not used by Geant and is simply passed through to the output tables.

- **start_vertex** and **stop_vertex** are the vertex_ids of the vertices where the track is born and dies. If the track does not die within the generated event, **stop_vertex = 0**. If **start_vertex** is not specified, all tracks in the event are assigned to a single primary vertex at the origin.

- **eg_pid** is an integer code to be used by the event generator giving the particle type using its own scheme, possibly different from GSTAR. It is not used by GSTAR and is simply passed through to the output tables.

- Either no vertex should be specified, in which case all tracks are assigned to a single primary vertex at the origin, or all primary and secondary vertices in the event cascade described must be specified. Each vertex (primary or secondary) is specified by a line

  VERTEX: x y z t vertex_id process parent_track n_daughters

  format: ('VERTEX:', 4(1x,g12.5), 4(1x,i6))

  where all fields are mandatory, if the **VERTEX** line is present.

  - **x,y,z,t** is the space-time coordinate of the vertex. This can be on the fermi length scale, the meter length scale, or anything in between.

  - **vertex_id** is a integer label unique within the event. The **start_vertex** and **stop_vertex** fields of the **TRACK** line refer to this field.

  - **process** is an arbitrary label specified by the event generator to denote the physical process that generated this vertex. It is not used by Geant and is simply passed to the output tables.

- **parent_track** is the track_id of the parent track of the vertex, i.e. the one that was killed at the vertex.

- **n_daughters** is the number of daughter tracks produced by the vertex.

- Comment lines: All lines not beginning with the strings **EVENT:**, **TRACK:**, **VERTEX:** or **GENER:** are ignored, and can be used freely to comment the file.

6.2 Event Generation

A Fortran shell (txgen_shell.F), Makefile and example applications for generating ASCII text format events are given in the directories under "gstar_lib-/txgen".

(i) generate_deltas.F: this file generates "generic" Delta resonances distributed uniformly about midrapidity with an exponential $p_\perp$ spectrum and decays them with equal probability into any combination of a nucleon and pion. (Note that we are not advocating a high net baryon density at midrapidity at RHIC - this example is only for illustrative purposes). The "Delta" is of course not known to GSTAR and therefore has ge_pid=0, with eg_pid arbitrarily assigned to 200. All vertices in this example are coincident with the origin, though they are distinguished by their labels. Following is the output of two events, each consisting of one primary vertex producing 5 Deltas, each of which decays, giving 6 vertices in total per event. For convenience, the shell has inserted comment lines which delimit the various fields:

```
******** Next Event ********
*         label_ ntrack nvert_
***
EVENT:         1     15      6

*** Vertices for this event ***
*        _____x______ _____y______ _____z______ _____t______ __id__ _proc_ parent ndaugh
***
VERTEX:  0.00000E+00  0.00000E+00  0.00000E+00  0.00000E+00       1      1      0      5
VERTEX:  0.00000E+00  0.00000E+00  0.00000E+00  0.00000E+00       2      5      1      2
VERTEX:  0.00000E+00  0.00000E+00  0.00000E+00  0.00000E+00       3      5      2      2
VERTEX:  0.00000E+00  0.00000E+00  0.00000E+00  0.00000E+00       4      5      3      2
VERTEX:  0.00000E+00  0.00000E+00  0.00000E+00  0.00000E+00       5      5      4      2
```

```
VERTEX:  0.00000E+00  0.00000E+00  0.00000E+00  0.00000E+00      6      5      5      2

*** Tracks for this event ***
*      ge_pid _____px_____ _____py_____ _____pz_____  __id__ startv stopv_ eg_pid
***
TRACK:      0   1.1248       1.1614       4.7842         1      1      2    200
TRACK:      0  0.47881      0.15308      -1.9919         2      1      3    200
TRACK:      0  0.29735      0.47014       2.6774         3      1      4    200
TRACK:      0 -0.23154      0.82929E-01  0.15353         4      1      5    200
TRACK:      0 -0.53841E-01  0.12885       1.3042         5      1      6    200
TRACK:     14  0.69150      0.72109       2.8522         6      2      0      0
TRACK:      8  0.43333      0.44035       1.9320         7      2      0      0
TRACK:     13  0.38883      0.16251      -1.1790         8      3      0      0
TRACK:      8  0.89979E-01 -0.94307E-02 -0.81287         9      3      0      0
TRACK:     14  0.24593      0.41839       2.6295        10      4      0      0
TRACK:      9  0.51424E-01  0.51745E-01  0.47914E-01    11      4      0      0
TRACK:     13 -0.14784      0.83291E-02 -0.98111E-01    12      5      0      0
TRACK:      7 -0.83698E-01  0.74600E-01  0.25164        13      5      0      0
TRACK:     13 -0.92994E-01  0.24197      0.79523        14      6      0      0
TRACK:      7  0.39153E-01 -0.11311      0.50895        15      6      0      0


******** Next Event ********
*         label_ ntrack nvert_
***
EVENT:         2     15      6

*** Vertices for this event ***
*       _____x______ _____y______ _____z______ _____t______ __id__ _proc_ parent ndaugh
***
VERTEX:  0.00000E+00  0.00000E+00  0.00000E+00  0.00000E+00      1      1      0      5
VERTEX:  0.00000E+00  0.00000E+00  0.00000E+00  0.00000E+00      2      5      1      2
VERTEX:  0.00000E+00  0.00000E+00  0.00000E+00  0.00000E+00      3      5      2      2
VERTEX:  0.00000E+00  0.00000E+00  0.00000E+00  0.00000E+00      4      5      3      2
VERTEX:  0.00000E+00  0.00000E+00  0.00000E+00  0.00000E+00      5      5      4      2
VERTEX:  0.00000E+00  0.00000E+00  0.00000E+00  0.00000E+00      6      5      5      2

*** Tracks for this event ***
*      ge_pid _____px_____ _____py_____ _____pz_____  __id__ startv stopv_ eg_pid
***
TRACK:      0  0.25731       1.4876      -4.5489         1      1      2    200
TRACK:      0  0.67651     -0.37136     -0.46025E-01     2      1      3    200
TRACK:      0  0.82671      0.58310     -0.57636         3      1      4    200
TRACK:      0 -0.27695      0.15733      0.71111         4      1      5    200
```

```
TRACK:       0 -0.39740     -0.77610        2.1639          5       1       6     200
TRACK:      13 -0.52344E-02  0.98295       -3.2959          6       2       0       0
TRACK:       8  0.26254      0.50463       -1.2530          7       2       0       0
TRACK:      14  0.47682     -0.47320     -0.16051           8       3       0       0
TRACK:       7  0.19969      0.10184      0.11449           9       3       0       0
TRACK:      14  0.58651      0.39117     -0.20438          10       4       0       0
TRACK:       8  0.24021      0.19194     -0.37198          11       4       0       0
TRACK:      13 -0.20064      0.11230      0.29789          12       5       0       0
TRACK:       7 -0.76311E-01  0.45034E-01  0.41322          13       5       0       0
TRACK:      13 -0.16361     -0.76427        1.8961         14       6       0       0
TRACK:       8 -0.23379     -0.11835E-01  0.26776          15       6       0       0


******** End of File ********
EVENT:      -999      0      0
```

(ii) generate_laserinos.F: This code generates a pattern of laser tracks consisting of "laserino" particles (currently ge_pid=170), which do not interact with matter or the magnetic field but nevertheless generate hits in the TPC. The generated pattern is not supposed to represent the actual STAR laser configuration, but again is for illustrative purposes only. In this case there are many primary vertices, each producing a few daughter tracks which are not decayed.

7 egz: Event Generators

See documentation for the egz package on the WWW under

`http://rsgi01.rhic.bnl.gov/star/starlib/doc/www/ssd/egz_l/egz.html`

or

`star->star_computing->software->package_docs->sim->egz`

8 Setting Up and Running GSTAR

A GSTAR run requires several components to be set up:

- environment variables specifying the names of the namelist files for all subsystems

- environment variable GSTAR_CARDS specifying the name of the file containing the Geant run-time control cards
- the contents of the run card file: this specifies which sensitive detectors to store hits in, which physics processes to turn on or off (Geant standard), plus a set of cards defined specifically in GSTAR.

8.1 Environment Variables

As an example, following is the file gstar_lib/gstar/exa/setup_gstar:

```
#!/bin/csh
#
# Define geometry environment variables.
#
setenv GEANT_LIB /afs/rhic/star/starlib/dev/sim/geant
setenv GSTAR_NML /afs/rhic/star/starlib/dev/sim/geant/gstar_geo/nml

setenv CAV_NML $GSTAR_NML/cave_v1.nml
setenv TPC_NML $GSTAR_NML/tpc_v1.nml
setenv SVT_NML $GSTAR_NML/null.nml
setenv PID_NML $GSTAR_NML/null.nml
setenv CTF_NML $GSTAR_NML/null.nml
setenv MAG_NML $GSTAR_NML/null.nml
setenv EMC_NML $GSTAR_NML/null.nml
setenv VPD_NML $GSTAR_NML/null.nml
setenv XTP_NML $GSTAR_NML/null.nml
setenv MLT_NML $GSTAR_NML/null.nml
setenv XX1_NML $GSTAR_NML/null.nml
setenv XX2_NML $GSTAR_NML/null.nml
#
# define run cards
#
setenv GSTAR_CARDS /afs/rhic/star/starlib/dev/sim/geant/gstar/exa/phys_off.gcards
#
#
# environment variables for GSTAR
#
setenv GSTAR_RUN 1
setenv GSTAR_NEVENT 2
```

An environment variable containing the file name of the namelist file of each subsystem is defined, as well as GSTAR_CARDS specifying the gcards file, used for setting physics flags, etc. (section 8.2). The "subsystems" XX1 and XX2 are hooks to test subdetectors for development work (not yet implemented).

The environment variables GSTAR_RUN and GSTAR_NEVENT are optional and can be used in a production environment to communicate from the shell script to the output tables the run number and number of events actually ordered. GSTAR checks that they are numbers, but their values are otherwise not used but simply passed along.

8.2 GSTAR Physics Control Cards

Following is the current file phys_off.gcards:

```
SETS 'TPC '
BMAP 0
FIEL 5.
VFLG 0
VXYZ 0. 0. 0.
VSGZ 0.
VSGT 0.
DCAY 1
ANNI 0
BREM 0
COMP 0
HADR 0
MUNU 0
PAIR 0
PFIS 0
PHOT 0
RAYL 0
LOSS 1
DRAY 0
MULS 1
STOP
```

The cards down to the line VSGT are defined specifically for GSTAR. The remainder are standard Geant physics control cards, documented in the

Geant v3.21 manual, section BASE040. In this case, all physics processes but decays, energy loss and multiple scattering have been turned off.

Following is the description of all cards currently defined with GSTAR. The card names are given in capital letters, the associated Fortran variable in the GSTAR code and its type are given in italics.:

- SKIP *n_skip_event n_skip_subevents n_skip_subran1 n_skip_subran2 (int)*

  Skip n_skip_event events in the input file before starting Geant processing. Default=0. For remaining fields (skipping subevents), see description of GEANT command line **user/skip** in section 8.3.

- BMAP *bmap (int)*

  bmap = 0 → use constant field. Otherwise, use field map. (Field map not yet implemented). Default = 0.

- BRAD *b_radius (real)*

  Maximum radius of field for bmap = 0. Default = 265 (cm.)

- FIEL *bfield (real)*

  Field value for bmap = 0. Default = 5. (kG)

- SCND *i_flag_secondaries (int)*

  i_flag_secondaries = 0 → ignore Geant-generated secondaries;

  i_flag_secondaries = 1 → put onto jstak but do not write to output file;

  i_flag_secondaries = 2 (default) → put onto jstak and write to output file

- VFLG *i_flag_prim_vert (int)*

  i_flag_prim_vert = 0 (default) → primary vertex offset = (0.,0.,0.)

  i_flag_prim_vert = 1 → primary vertex offset gaussian distributed about (0.,0.,0.) using sigma_prim_vert_xy and sigma_prim_vert_z

  i_flag_prim_vert = 2 → primary vertex offset gaussian distributed about (x,y,z) using sigma_prim_vert_xy and sigma_prim_vert_z

- VSGT *sigma_prim_vert_xy (real)*

  Sigma of transverse distribution of interaction points. Default = 0.

- VSGZ *sigma_prim_vert_z (real)*

  Sigma of longitudinal distribution of interaction points. Default = 0. ("diamond length")

- VXYZ *prim_vert(3) (real)*

  Coordinates of mean position of interaction points. Default = (0.,0.,0.)

- NSUB *n_track_per_subevt (int)*

  Number of "final state" tracks per Geant subevent. Default = 200.

- GDBG *i_flag_geo_debug (int)*

  If non-zero, all defined materials, tracking media and volumes are printed after initialization. Mostly for use by developers of geometry. User can check consistency by hand. Default = 0.

- SHDO *make_shadow (int)*

  If hadronic physics is turned off (HADR 0 in .gcards file), tracks can pass through dense material without interacting. Acceptances calculated with hadronic physics off may be unphysical, since the dense materials will not cast a shadow. If hadronic physics is off and make_shadow=1 (default), tracking thresholds in dense materials will be set very high, thereby restoring their shadows. This mechanism is disabled if hadronic physics is turned on. If make_shadow=0, normal tracking thresholds are used, a warning is printed upon startup, and dense media do not cast shadows.

- CUTE *cutele_gas (int)*

  If cutele_gas is positive, sets lower bound on track energy for tracking electrons in sensitive gases. If negative (default), Geant default parameter of 1 MeV is used. Lowest recommended value is 10 KeV (i.e. cutele_gas=.00001, remember that units are GeV). Note that this does not alter electron tracking cuts in any media other than sensitive gases. An accurate calculation of delta electrons ejected from surrounding materials into the sensitive detectors requires further study and discussion.

The primary vertex offset (VFLG) is an offset applied in GSTAR to the event generator coordinate system for a given event. For instance, if the event generator produces all primary vertices at (0.,0.,0.) in its own coordinate system, the primary vertex offset will be the coordinates of the primary vertices of the event in the GSTAR coordinate system.

8.3 Running GSTAR

Once these files are set up and the environment variables defined, run the executable **gstar_lib/gstar/bin/gstar_PP**. For workstation type, returning 0 does not create a graphics window, returning default will create a standard HIGZ window, and returning 999 or 'm' will create a PAW++-like interface. Refer to the Kumacs in section 9 for opening input and output files, triggering events, and drawing the detector.

Following are GSTAR-specific commands that can be issued from the GEANT command line:

- event input:

  ```
  GEANT> user/input type file1 (file2)
  ```

  type denotes the type of file (fzd or text), and takes the values FZ (event generator ZEBRA format), TX (new ASCII text format), TX-OLD (old ASCII text format) and FZTX (two input files, the first of which is FZ and the second is TX)

  file1 denotes the filename of the file (or the FZ file in the case of FZTX input)

  file2 denotes the name of the TX file in the case of FZTX input

- event output:

  ```
  GEANT> user/output action filename
  ```

  action takes the values O (open output file) and C (close output file)

  filename is the name of the file in the case that action = O.

- Flat phasespace event generator:

  ```
  GEANT> user/phasespace id ylow yhigh ptlow pthigh ntrack
  ```

Phasespace sets up the generation of events consisting of a single type of particle having geant pid = **id**, within the rapidity window [**ylow, yhigh**] and $p_\perp$ window [**ptlow, pthigh**], uniformly distributed with **ntrack** tracks per event, originating at the primary vertex. Azimuthal angle is distributed uniformly over $[0,\ 2\pi]$.

If **yhigh** < **ylow**, then **yhigh** is set equal to **ylow**; if **pthigh** < **ptlow**, then **pthigh** is set equal to **ptlow**. In this way, **ntrack** identical tracks (except for azimuthal angle) can be generated per event.

- Flat momentum space event generator:

```
GEANT> user/momentumbin id pxlow pxhigh pylow pyhigh
                                          pzlow pzhigh ntrack
```

Momentumbin sets up the generation of events consisting of a single type of particle having geant pid = **id**, within the 3-momentum bin defined by the px, py and pz limits, uniformly distributed with **ntrack** tracks per event, originating at the primary vertex.

If the upper limit of a component is less than its lower limit, the upper limit is set equal to the lower limit. If this is true for all three components, **ntrack** identical tracks will be generated per event.

- Skipping of events from input file:

```
GEANT> user/skip nevent nsubevent subran1 subran2
```

The first **nevent** events of the input file are skipped. The first event tracked by GSTAR is event number **nevent**+1. This is of use in a production environment: because of the large magnification of event size from input to output in GSTAR/G2T relative to the event generator event, only a small fraction of the events in an input file can be processed in one GSTAR run.

This mechanism is also of use for debugging. It will happen that a problem will occur after many hours (or days) of running. To reproduce the problem, it is undesirable to repeat the entire run, or even the entire event in which the problem occurred (which may require a cpu-hour or more). If **nsubevent**>0, GSTAR will skip the first **nsubevent** subevents of event **nevent**+1, starting to process with random seeds **subran1, subran2**. In this way, the state of GSTAR is precisely reproducible at the start of any subevent.

- Spatially uniform decay of parent to given decay branch

```
GEANT> user/udecay pidparent rin rout pid1 pid2 (pid3)
```

Udecay sets up the decay of a given particle type specified by Geant pid **pidparent** into a given 2- or 3-body decay mode specified by daughter **pid1, pid2** and optionally **pid3**. Decay vertices for this parent will be uniformly distributed along the parent's trajectory between radial limits **rin** and **rout**.

This is of use in studying decay modes which occur a measurable distance from the primary interaction. Often the efficiency of the Monte Carlo is significantly hampered due to the exponential loss of parents before they reach the fiducial volume for the decay vertex. **Udecay** "removes" this exponential component, whose weight for producing a decay vertex at a given path length can be put in by hand post facto.

Technically, a new particle is defined by GSTAR with **pidnew** = **pidparent**+200, with all the properties of **pidparent** but infinite lifetime. Whenever either **pidnew** or **pidparent** is encountered in the input event stream (from any of the GSTAR event sources), it is set to **pidnew** and a decay vertex position is explicitly calculated by GSTAR, bypassing the standard GEANT decay routines (though utilizing the GEANT CM decay kinematics routines). The newly defined particle appears in the GEANT particle definition table but its defined decay mode does not currently appear in the decay mode table (for minor technical reasons).

Note that the three-body decay routine within GEANT (used by udecay) assumes scalar particles (no spin effects).

The parent must have taken at least three tracking steps in GEANT before decaying. Because of this restriction, it is possible that this mechanism as currently designed will not work for decays on the scale of a few hundred microns (e.g. D mesons). This has not yet been investigated.

- Define new particle

```
GEANT> user/spartstar ipart napar itrtyp amass charge tlife
ubuf nwbuf bratio mode
```

SPARTSTAR has the same functionality as the standard GEANT command SPART with the significant exception that it works (bug fix). Note that **ubuf**, **bratio** and **mode** are vectors (in the PAW sense). Following is a kumac that will define the phi meson as pid 75 and define its decay modes. Recall that GEANT imposes a maximum of six decay modes for any particle, thus for convenience we lump the $\rho\pi$ branch into the $\pi^+\pi^\circ\pi^-$ branch (yes, it's wrong) so that the e^+e^- branch can be included. See GEANT manual, sections CONS-300 and CONS-310 for more information.

```
vector/delete *
vector/create ubuf(1) R 0.
vector/create bratio(6) R .491 .343 .154 .0128 .00131 .0003
vector/create mode(6) I 1112 1016 70809 1701 701 203
spartstar 75 'phi' 4 1019.4 0. 1.5e-22 ubuf 0 bratio mode
print part 75
gfdk 75
```

- Print decay modes

```
GEANT> user/GFDK ipart
```

GFDK prints all defined decay modes defined for parent particle id ipart.

9 Shell Scripts and Kumacs

In this section we describe some shell scripts and Kumacs for setting up and running GSTAR. All files are found in the directory $GEANT_LIB/gstar/exa. It is recommended that the user copy these over to her/his private area to modify them as needed. All file references are absolute, so that no modification is necessary for these to run, except that the user should set up a local directory ../fzd for the ZEBRA output. For convenience, the user should also make the following soft link in her/his private area:

```
$ ln -s /afs/rhic/star/starlib/dev/sim/geant/gstar/bin bin
```

Then GSTAR can be started simply by typing

```
$ bin/gstar_PP
```

- setting of environment variables: the file **setup_gstar** defines all namelist environment variables as well as the .gcards filename. Its content is listed in section 8.
- drawing Kumacs: the files **draw_star_wireframe**, **draw_star_low_resolution** and **draw_star_high_resolution** generate colour perspective drawings of the subsystem envelopes of STAR. Wireframe draws only the envelope boundaries, whereas low-resolution and high-resolution fill the areas with colour. High-resolution produces the best quality figures but is very slow over the network. Colour scheme: TPCs are red, SVT is green, Calorimeter is violet, all electronics are light blue, magnet is dark blue, Central Trigger Barrel and VPD are yellow, and beam pipe is black. The wireframe drawing is shown in Fig. 3.
- Geometry diagnostics: An example of how to use the diagnostics Kumac produced in association with the TPC namelist file is given in tpc_diagnostics.kumac. The lower left quadrant of the TPC endcap is drawn and then the diagnostic Kumac is run. Every volume (almost) given in Fig. 4 is drawn after the user has been prompted (simply given a carriage return at the prompt). Clearly not all volumes are distinguishable at this scale. A full diagnostic would require several different views at different magnification.
- laser events: run Kumac run_laserinos.kumac. This needs a soft link to the event file gstar_lib/txgen/out/laser_event.txt, which was generated by gstar_lib/txgen/src/generate_laserinos.F. Output is written to the local directory. Note that the geometry of the laser sources in this file has NOTHING WHATSOEVER TO DO WITH THE STAR LASER SYSTEM!! It was made only for illustrative purposes. Based upon this example, implementation of the correct laser beams is a simple job. After the tracks are drawn, the hits in the TPC are redrawn, prompted by the user.
- running an event generator file: run_venus.kumac runs one central Au-Au event and writes fzd output to the local directory.

- defining new particles and decay modes: see define_phi.kumac.
- GSTAR internal event generators, generating spatially uniform decay: see run_udecay.kumac.

10 Running g2t

Synopsis:

g2t [-i geant_input_file] [-o dsl_output_file] [-n no_of_events]

g2t reads ZEBRA files with Geant output and fills the tables described above (see. section 5). The complete web of mutual pointers is resolved as a service for the user. The output is in the new DSL format (see on the WWW:)

```
star->star_computing->software->package_docs->sys->dsl
```

Example: g2t can be started by typing e.g.

```
$ $GEANT_LIB/g2t/bin/g2t -i myfile.fz -o myfile.dsl -n 10
```

for the conversion of 10 events in the GSTAR ZEBRA output file myfile.fz.

- Input file: can be given either as a command line option, or by creating a symbolic link g2t.input to the input file:

  ```
  $ ln -s geant_input_file g2t.input
  ```

- Output file: can be given either as a command line option, or by creating a symbolic link g2t.output to the input file:

  ```
  $ ln -s dsl_output_file g2t.output
  ```

- Number of events to process: can be given either as a command line option, or will be prompted for by the program.
- Running as a pipe or filter: To prevent the overhead of storing a large ZEBRA file and later converting it into a DSL file, g2t can be run via a named pipe or FIFO. It cannot be run in the usual UNIX pipe mode due to limitations of the ZEBRA input. The FIFO can be set up in the following way:

```
$ mkfifo /usr/tmp/g2t.fifo
$ g2t -i /usr/tmp/g2t.fifo -o my_output_file.dsl -n 10 &
$ ln -s /usr/tmp/g2t.fifo geant_zebra.data
$ gstar_PP .... (run gstar_PP according to section 8)
$ rm /usr/tmp/g2t.fifo
```

N.B.: do not try to create a FIFO in an AFS area. Since AFS buffers files, this leads to undesirable results.

- Inspection of DSL files: as soon as the g2t package and its output tables are finalized, they can be simply read into TAS. Furthermore, there will be the possibility to read any DSL table into TAS sometime in the future. For the time being, a translation utility provided by C. Tull should be used:

  ```
  $ translate_xdf2rzd my_output_file.dsl
  ```

  This command has to be run from the same directory the .dsl file resides in. It will produce a file my_output_file.rzd, containing one CWNtuple for each table and event, which can then be easily inspected via PAW. This command is available at present on aix, hpux, and sun4os5. It does **not** work on SGI due to a defect in the STAR CERN libraries.

11 Figures

List of Figures

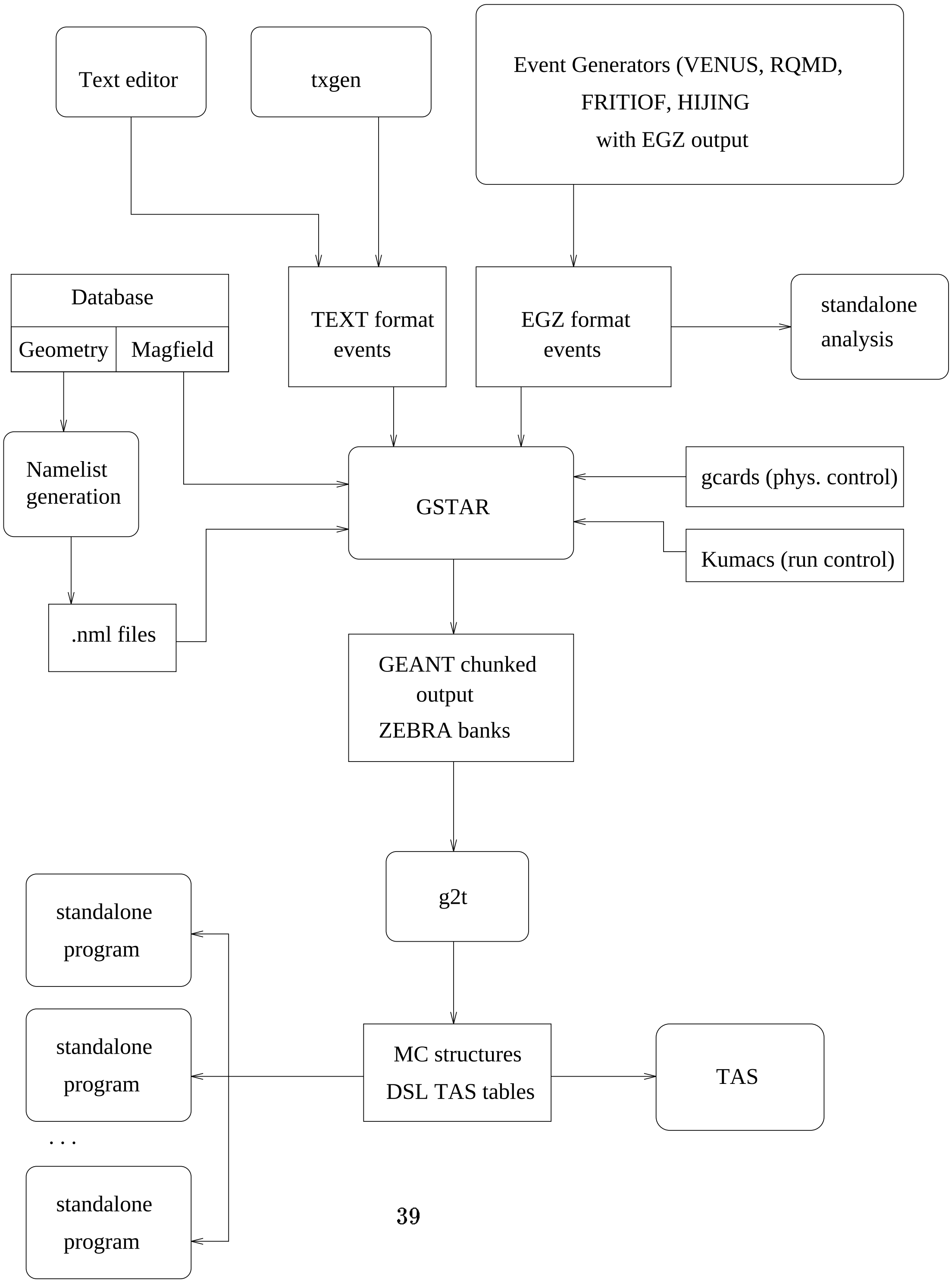

Figure 1: Schematic logical flow within the GSTAR chain

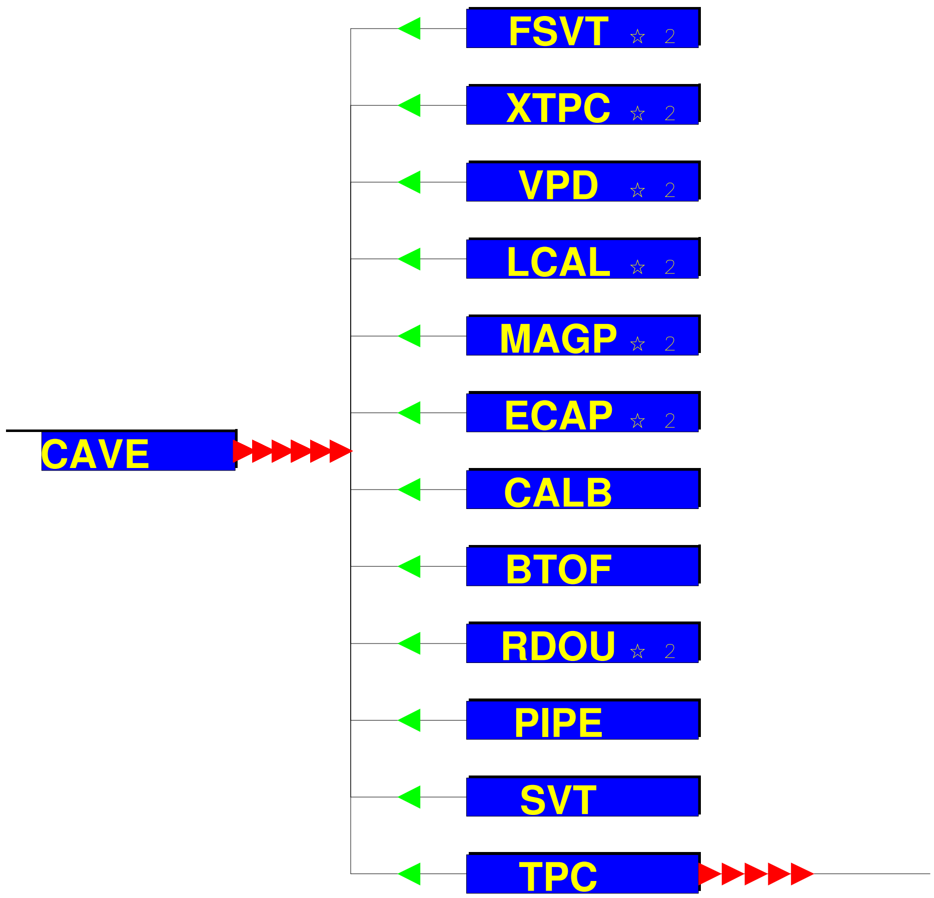

Figure 2: Logical tree of Geant volumes inside the CAVE

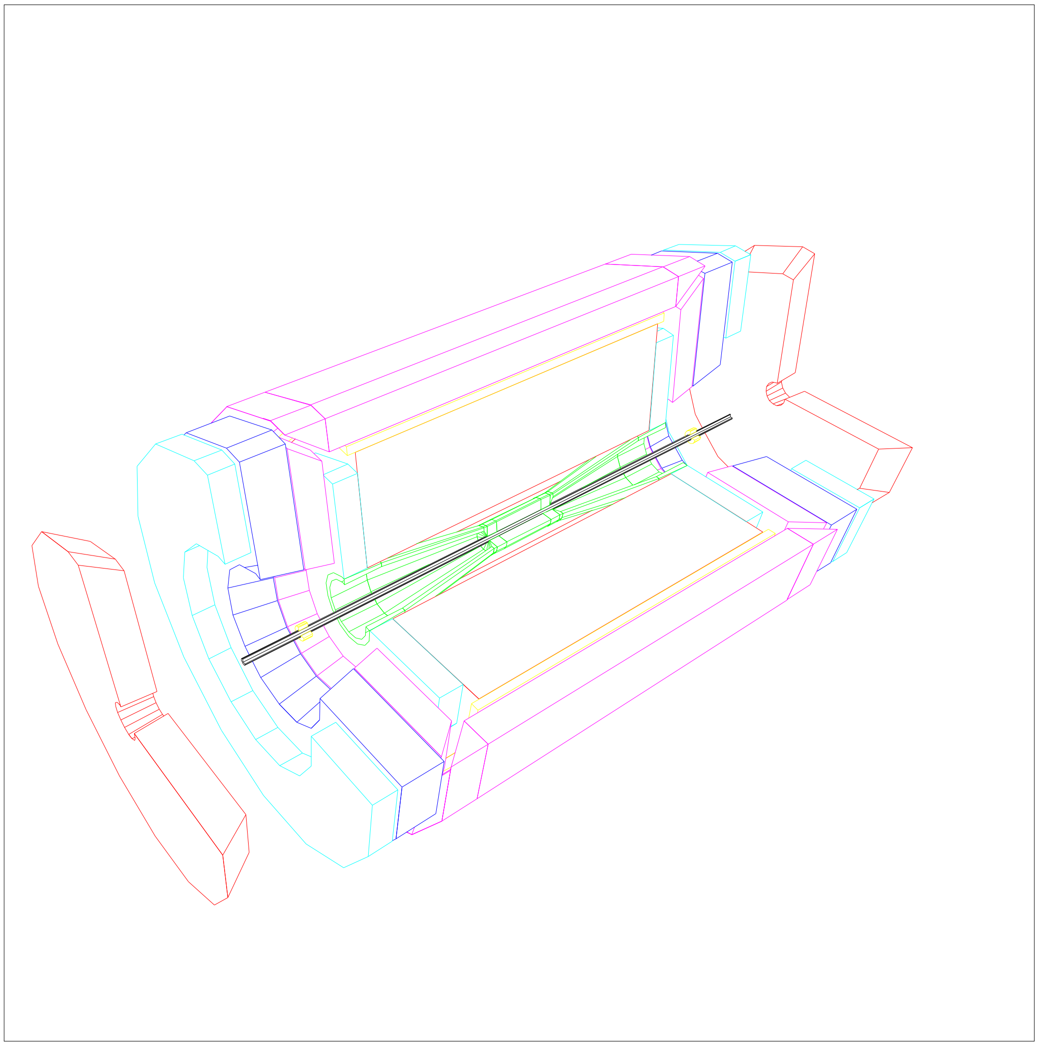

Figure 3: Wireframe drawing of the STAR envelopes

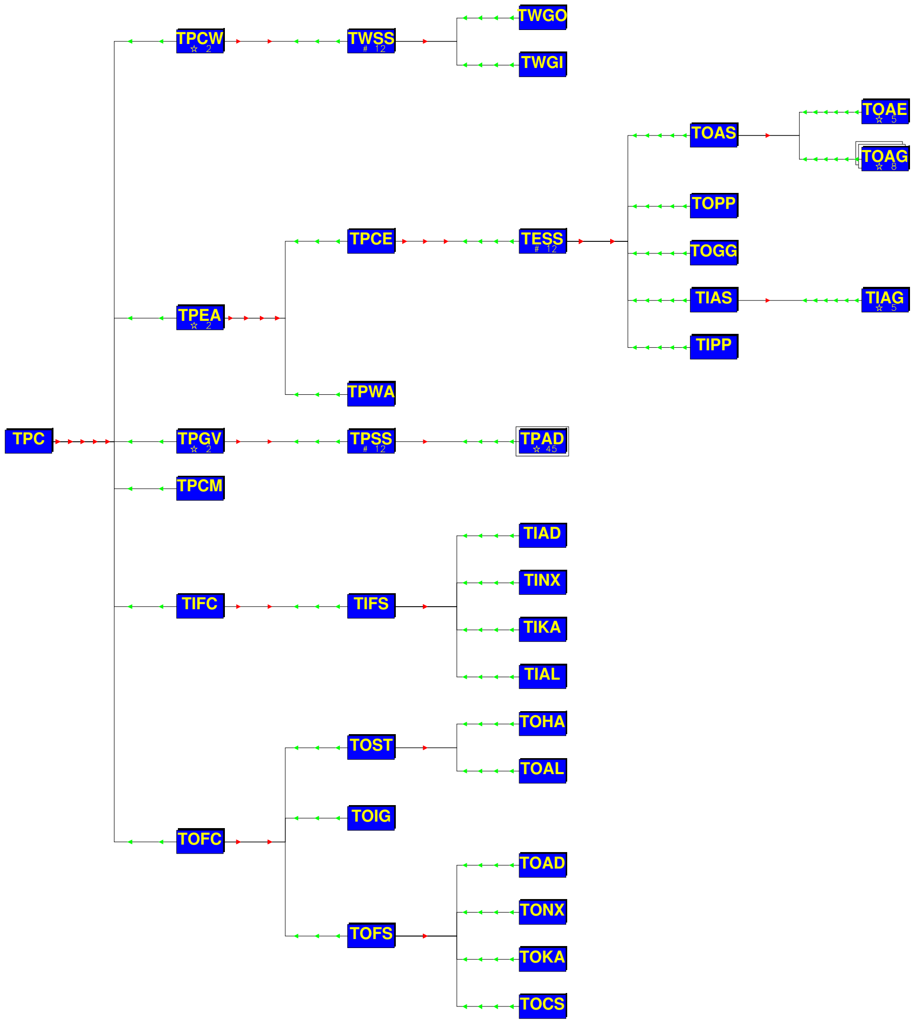

Figure 4: Logical tree of Geant volumes inside the TPC

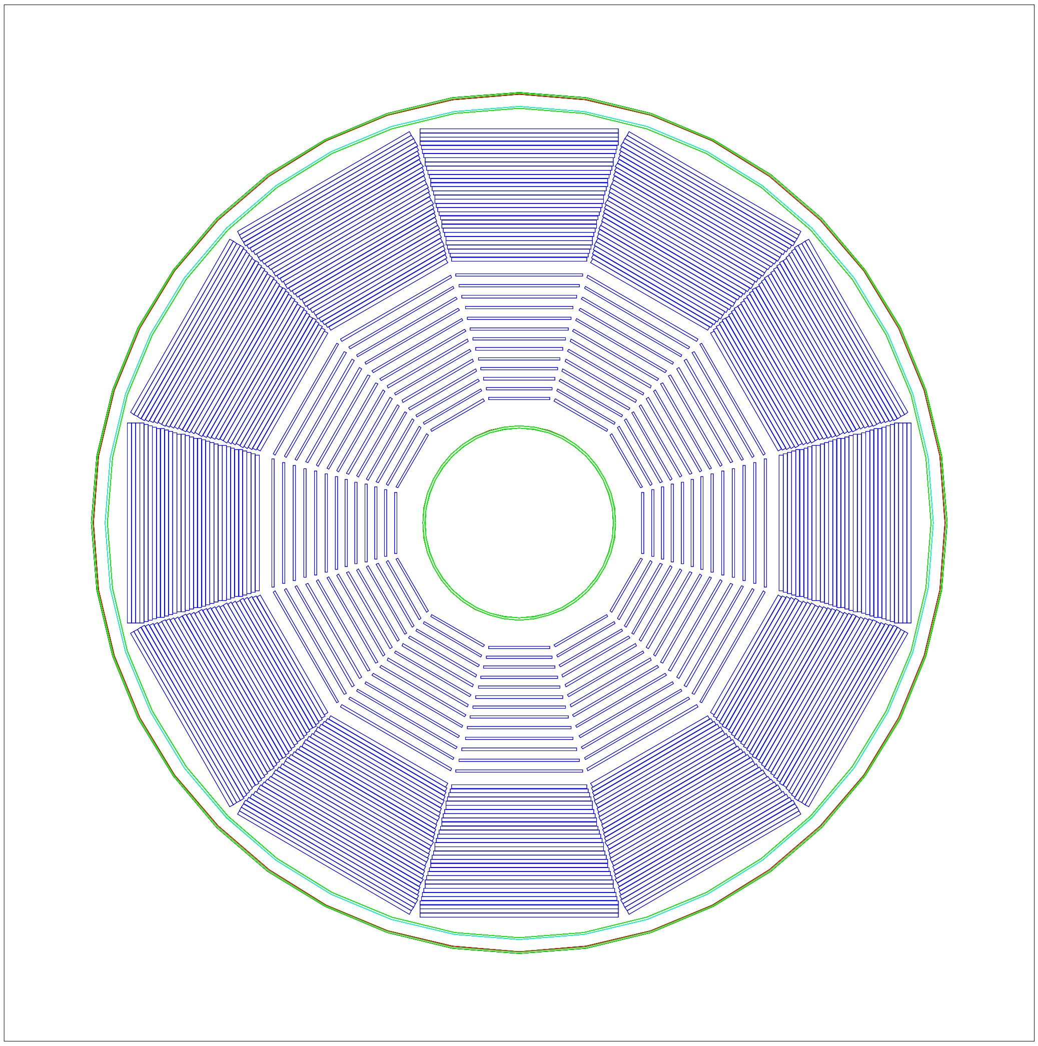

Figure 5: Padrow volumes and field cages, TPC endcap view.

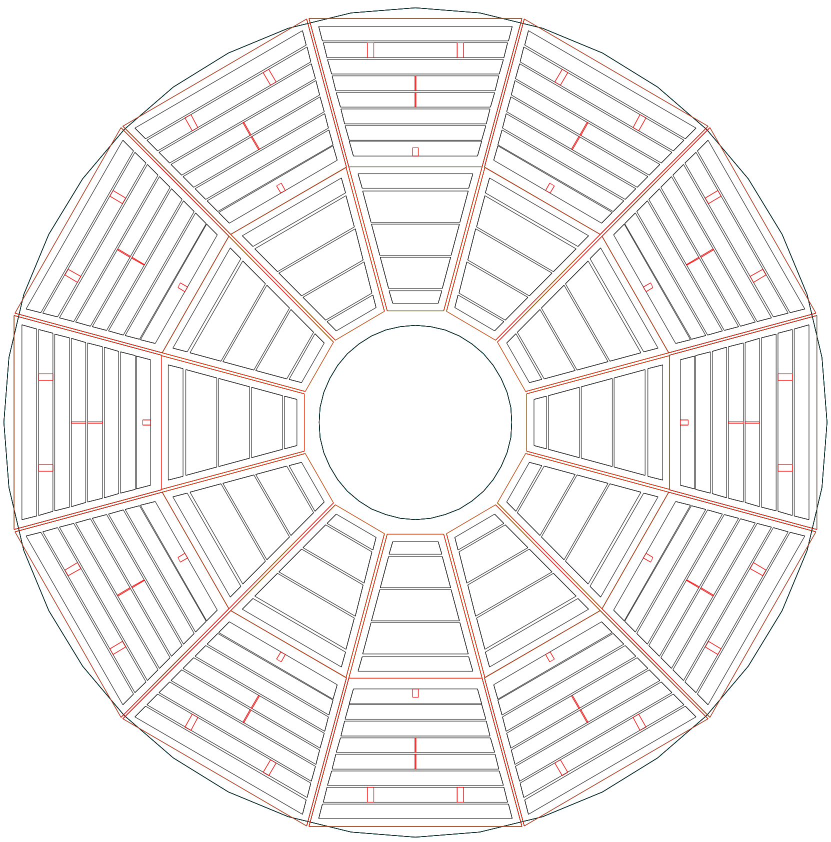

Figure 6: Aluminum sector structures of TPC endcap.

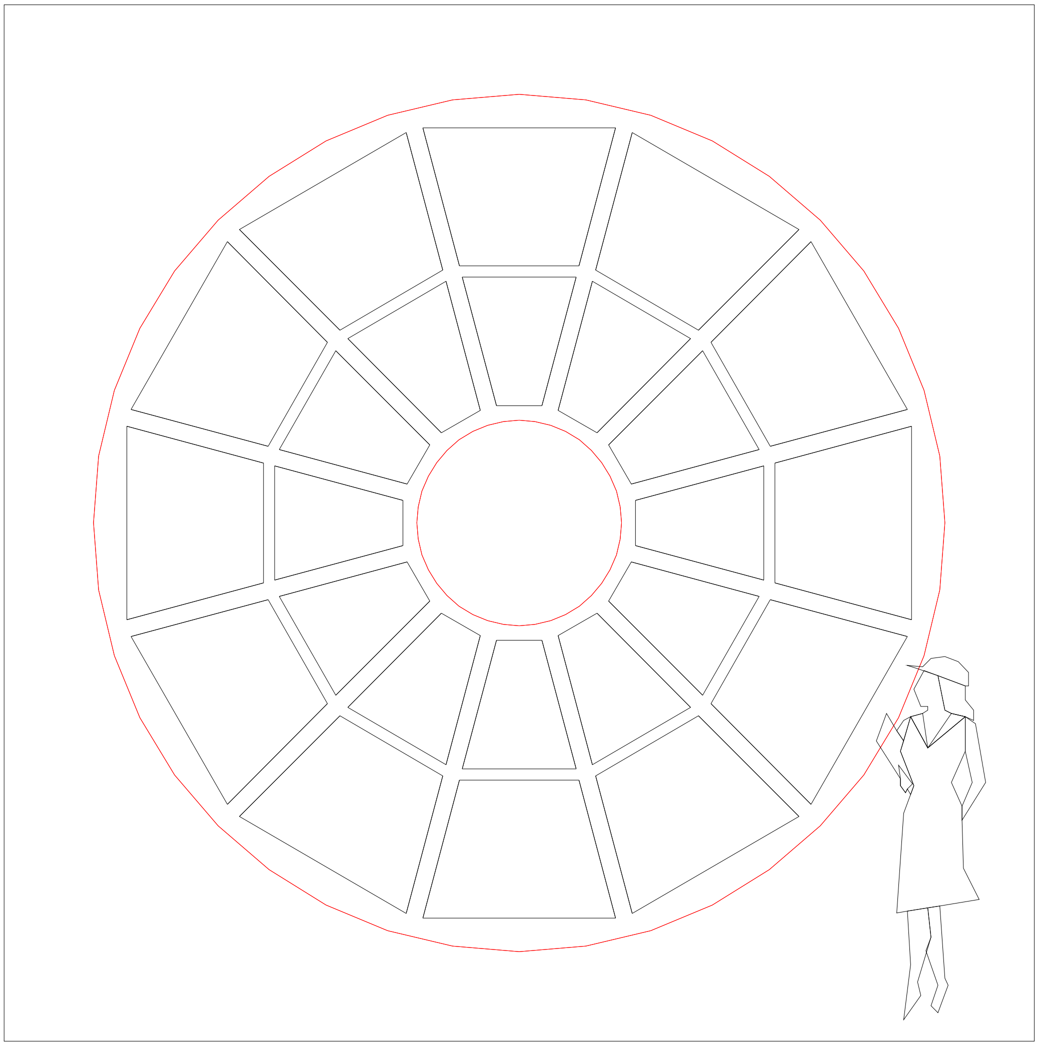

Figure 7: Aluminum support wheel of TPC endcap.

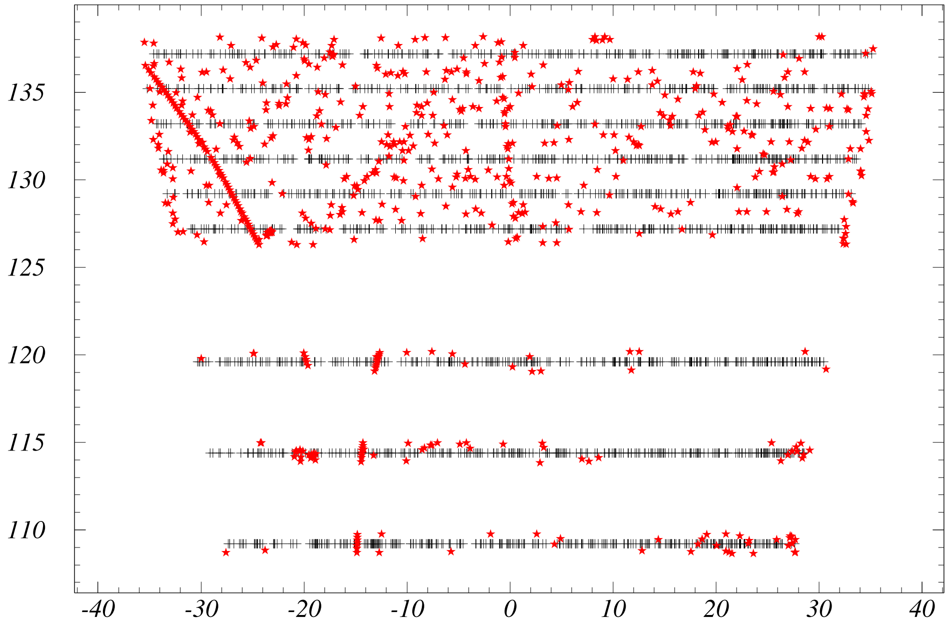

Figure 8: TPC hits for one Au-Au event on a few padrows near the middle of sector 12. Tracks which traverse the complete padrow have their hits placed at the middle of the padrow. Tracks which enter only the top or bottom surface and either die within the padrow (interactions, decay) or exit through the sides (at the sector boundary, the endcap or the midplane), as well as delta electrons, have their hits at a position weighted by the trajectory within the padrow.