SLAC – PUB – 4787
November 1988
E/I

Simulating a Central Drift Chamber for a Large Solenoid Detector at the SSC, Using GEANT3*

ANDREA P. T. PALOUNEK

Stanford Linear Accelerator Center
Stanford University, Stanford, California, 94309

⋆ Work supported by the Department of Energy, contract DE – AC03 – 76SF00515.

1. Introduction

One possible difficulty with using conventional technology such as a drift chamber tracking device at the SSC is the worry that the event rate and multiplicity will simply swamp the detector and make it useless for interesting physics studies. The only way to answer such concerns rationally is to model a detector accurately and reconstruct interesting events after they have been overlaid with background events. This is what Gail G. Hanson set out to do in 1986, later with the help of Bogdan Niczyporuk. I joined the project in February 1988, shortly after Niczyporuk left to work at CEBAF. This note mainly describes the GEANT Monte Carlo program we used to model the tracking chamber, with a few comments about the GEANT environment at SLAC. Obviously, it mainly describes the programming efforts of other people.

The tracking chamber we have been modeling is part of a large solenoid detector[1] studied during the Berkeley Workshop in 1987. The design is along fairly conventional lines: it consists of a microvertex detector surrounded by first a central tracking chamber and then a calorimeter. These are placed in a 2 T magnetic field, with muon chambers, the coil flux return, and another set of muon chambers placed radially. It is not a proposal for a detector - simply an exploration of what a reasonable detector at the SSC might be.

GEANT was imported to SLAC by the SLD collaboration. SLD computer experts David Aston, John Brown, and Terry Reeves reconstructed the source from PAM files and installed it at SLAC. They did an immense amount of work making the system work and interfacing it to SLAC unified graphics. We of the SSC effort have taken their implementation of GEANT and have used it with very little modification. Members of SLD have continued to develop their GEANT, casting the program into the collaboration's IDA-JAZELLE environment. We have used the standard, CERN version with the data manager ZEBRA. However, the work of Aston et al. has been invaluable to this project; without it we would have had to put in much more work to get started. We are also very much indebted to their kind answers to our many questions while we were trying to understand GEANT.

This note describes GEANT in general, with examples from the SSC tracking chamber study. There is also a section on running GEANT at SLAC, again with examples from the SSC study. Finally, there is a short section on ZEBRA, the data management system used by GEANT.

2. The GEANT Detector Simulation System

GEANT[2] is a large general purpose system of detector description, simulation, and graphical representation tools which was developed explicitly for High Energy Physics experiments. First written nearly 15 years ago to model SPS experiments, it has been used by many collaborations in most of the HEP community's laboratories. The current version, GEANT3, was rewritten in 1983 and includes more emphasis on the description of complex geometries as well as more sophisticated modeling of calorimetry. It includes the General Hadronic and Electromagnetic Interaction Shower code GHEISHA[3] which in turn includes the electromagnetic shower package EGS[4] . GEANT3 is being used by many experiments at LEP, SPS, LEAR, HERA, and SLC, and continues to be receptive to the users from those collaborations.

GEANT3 is a collection of FORTRAN77 routines, grouped together into several modules. It relies extensively on CERN Library routines, including the data manager ZEBRA[5] . The modules each perform one particular class of functions and are:

- the GEOMetry package to describe the experimental setup;
- the CONStants package to keep the particle, material, and tracking medium parameters;
- the PHYSics package to model particle interactions with the matter in the detector;

- the KINEmatics package to generate simulated data from parameters in the standard physics PHYS package;
- the TRAcKing package to transport the resulting particles through the simulated detector, taking into account the various geometrical boundaries and the materials in the detectors;
- the HITS package to record the responses of the detector to the particles;
- the DRAWing package to draw the detector, the particle trajectories, and the hits;
- the I/O PAckage to record or retrieve events on an external device in a machine-independent way;
- the (X)INTeractive package to allow the GEANT3 user to edit many parameters and execute commands.

Many of these modules contain dummy and default user subroutines; these are called whenever the user may have application-dependent procedures to perform. For programming ease, the names of these routines always begin with the characters GU.

Units throughout the system are grams, centimeters, seconds, and kiloGauss.

The GEANT3 user is expected to provide the data to describe the detector, to code the relevant GU subroutines, to assemble the required modules and their attendant routines into a logical, executable program, and to provide data cards which control the execution of the program. Because most of the effort involved in creating a good GEANT3 program is in the geometry description of the detector, the following section describes the GEOMetry package in some detail and uses the DRAWing package to illustrate some of the tools available. Fortunately for GEANT3 programmers, the DRAWing package allows the user to check easily that the detector description, in principle an easy task in coordinate geometry, does in fact correspond to the physical attributes of the detector.

2.1 The Geometry Package: GEOM

The general structure of the geometry description package is a nested tree form. By convention, an overall supervolume is defined to contain the entire detector. Within this supervolume are various volumes, usually each corresponding to a subdetector such as a central calorimeter or drift chamber. These volumes are defined with a name, a shape, and necessary parameters such as dimensions, material physical properties, tracking properties such as the magnetic field, and a local coordinate system. Then they are placed in the supervolume with the correct copy number, position, and rotation. The GEANT system allows for thirteen shapes, including tubes, boxes, spheres, trapezoids, and cones. Each shape has its own defined coordinate axes.

Each volume can in turn be partitioned into subvolumes such as the hadronic and electromagnetic sections of the calorimeter or drift chamber superlayers, and these placed appropriately. Multiple copies of a (sub)volume can be placed in the same or different volumes.

Each (sub)volume can be segmented equally into divisions, such as calorimeter modules or drift chamber layers. These divisions can be created along any of the three axes of the mother volume - whether cartesian, cylindrical, or spherical. A given volume and all its properties can be repeated several times in a manner which minimises the information storage. This is particularly useful in the very common case of several nearly identical subdetectors within one large detector.

This nesting can be repeated up to 15 times to provide the desired level of description - down to the O-rings and bolts, if necessary. Each detector level has its own name or set of names, depending on how it was created. There are user and system volume numbers at every level to distinguish several nearly identical volumes, if necessary. Each level may have its own coordinate system.

This structure has several advantages. It follows the usual conceptual method of starting with the whole and subdividing each part until the desired level of detail is reached. It allows for quick, easy reference to each detector part as, say, the eighth wire of the first superlayer of the third module of the drift chamber. Its recursive definitions make modification of the detector description relatively easy.

The VOLUME definition

The unique, initial volume which encompasses the entire setup is defined with no location; the reference frame related to this volume is the master reference system or MARS.

In the simulation of central tracking for the SSC, the entire detector is called GLOB and contains five levels of description. The central drift chamber, which nearly fills the entire GLOB volume, is called CHAM. It consists of thirteen tubular modules called MD01 to MD13. Here, the 'MD' designates a module and the rest the module number, with module 1 closest to the center and module 13 outermost. The module radii and lengths in z are explicitly defined by the program. Each module contains one GEANT volume called a superlayer. Because volume names are limited to four characters by GEANT, the superlayers are designated by the letter 'S' and a three digit number. The first two digits refer to the module which contains the superlayer; the last digit refers to the superlayer number within the module. Since the current application has only one superlayer per module, the superlayers are called S011 to S131. Each superlayer has eight divisions or layers. GEANT requires that all divisions of a given volume have the same name, so the layers are called L011 to L131, in direct analogy to the superlayers. The layers, at the fifth and final level of detector description, are distinguished by their volume numbers only.

A representation of the structure, created using the interactive package and the command

```
        DTREE GLOB 0 10001
```

is in figure 1.

The geometrical definition is achieved with only a few GEANT calls. First, GLOB and CHAM are created with:

```
CALL GSVOLU('GLOB','TUBE',1,AVOLG,3,IVOL)
CALL GSVOLU('CHAM','TUBE',4,AVOLCH,3,IVOL)
CALL GSPOS('CHAM',1,'GLOB',0.,0.,0.,0,'ONLY')
```

This creates a tubular volume called GLOB which is made of material number 1, with shape parameters contained in the array AVOLG of three parameters (minimum radius, maximum radius, and half-length along z), and receives a volume number IVOL. It similarly creates a volume CHAM, then uses GSPOS to position CHAM copy 1 into GLOB, with the reference frames equal, and specifying that if a point is in CHAM it is in no other volume.

Then modules and superlayers are created by a loop over:

```
CALL GSVOLU(NAMESM,'TUBE',4,AVOLM,3,IVOL)
CALL GSPOS(NAMESM,1,'CHAM',0.,0.,0.,0.,'ONLY')
CALL GSVOLU(NAMESS,'TUBE',5,AVOLM,3,IVOL)
CALL GSPOS(NAMESS,1,NAMESM,0.,0.,0.,0.,'ONLY')
```

These, called thirteen times to create the thirteen modules and superlayers, create tubular volumes of the name kept in the variable NAMESM (names - module, or MD01 to MD13) which are made of material 4, and have shape parameters AVOLM(3). The NAMESM are placed as 'only' volumes into CHAM. There are also tubular volumes of names NAMESS, (names - superlayer, or S011 to S131) similarly created but out of medium 5, and placed into the modules NAMESM.

It would also be possible to position a number of modules with the same shape but different dimensions by defining a generic shape with an alternate call to GSVOLU:

```
CALL GSVOLU(NAMESM,'TUBE',4,AVOLM,0,IVOL)
```

(notice the fifth variable, the number of shape parameters, is set to zero) and use the subroutine GPOSP, which takes a copy of a named, previously created volume and places it at a specified location and rotation.

Finally, each superlayer is divided into layers:

```
    CALL GSDVN(NAMESL,NAMESS,NLAY(IMOD,ISLAY),1)
```

This specifies GEANT is to create division NAMESL (names - layer, or L011 to L131) from NAMESS, with NLAY slices or divisions along the 1-axis. NLAY is an array with dimensions the number of modules and the number of superlayers per module, currently set to 13 and 1, respectively. At the moment, NLAY is always 8. Notice all the layers created this way have the same name; they are distinguished by their volume numbers only.

The GEANT system allows for other sorts of divisions: the subroutine GS-DVT allows the user to specify a division STEP, rather than the number of divisions NDIV, as well as the medium number. There are plans for GSDVX which would allow for specification of a STEP as well as NDIV, along with the origin of the first cell and the tracking medium. This would create gaps at either end of the mother volume.

The data which drive the GEANT calls are kept in the user common block DCGEOM. This contains the number of modules, number of superlayers, minimum and maximum radii for modules, superlayers, and layers, the half-length of the modules along the beam direction, the number of sense wires in each layer of a superlayer, the cell width in a superlayer, the azimuthal angle of the arbitrarily chosen first wire in the layers, and the electron drift velocity:

```
      COMMON /DCGEOM/ NMOD, NSULAY(20), NLAY(20,20), RMMIN(20),
    * RMMAX(20), RSMIN(20,20), RSMAX(20,20), RLMIN(20,20,20),
    * RLMAX(20,20,20), XMLEN(20), NSWIRE(20,20),
    * SWIDTH(20,20), PHILAY(20,20,20), VD
```

To save enormously on processing time, GEANT does not know about the details of cell structure or wire placement. It is the SSC user who must keep track of the details of cell structure and wire position. While this saves tremendously on computing time, it costs in loss of these data when the detector geometry is written to tape or disk. An important improvement of the current system would save the more detailed data in a ZEBRA structure and record them with the GEANT geometrical information. The ability to input the DCGEOM data in a more user-friendly way, perhaps using a menu, would be another useful improvement.

Ordering and Closing the Detector Geometry

GEANT has a specific search order to find in which volume a particle is. This ordering may be specified explicitly, by frequency of entry, or geometrically. In the SSC tracking example, the search order is defined geometrically: that is, GEANT keeps track of the limits of each of the contents of a volume, and which contents are in each of the sections defined by the surrounding coordinates. This is specified with:

```
CALL GSORD('GLOB', 1)
```

This orders the entire detector GLOB along the 1-axis, radially.

After all volumes and positions and orderings have been defined, the geometrical and search data should be stored. This is done by:

```
CALL GGCLOS
```

2.2 The Physics Constants Package: CONS

After the geometrical representation of the detector is complete, the user must still define the materials which fill the detector volume. This material may be a pure material such as liquid argon or copper, or it may be a nonexistent average of several materials. This is useful when modeling a drift chamber, for example: a material can be defined with the average properties of the gas mix and wires, eliminating the need to model every wire separately. There is also a routine available to make mixtures or compounds from previously defined materials.

Defining the material

The subroutine GMATE will simply store necessary constants for sixteen standard materials, including Hydrogen, Lead, and Air.

The subroutine GSMATE is more versatile. Not only can the user define only those standard materials needed for the particular application, but also any other material or mix. The routine requires the user-defined material number and name, along with its atomic weight and number, density, radiation length, absorption length, and other (optional) user parameters.

For the SSC central tracking case, where the detector is surrounded by air and is a mixture of Mylar, glue, stainless steel wires, argon, and ethane, this is:

```
CALL GSMATE
    (15,'AIR$', 14.61, 7.3, 0.001205, 30423.24, 6750., 0,0)
CALL GSMATE
    (18,'STRAW$', 21.  7, 10.0, 0.023, 1370., 61.6, 0,0)
```

These define material 15 to be air, with the given atomic weight, atomic number, density, radiation length, absorption length, and no other parameters. Material 18, called 'straw', has the shown properties and is actually a weighted average of appropriate amounts of gas, Mylar, and wire. Material number 15 is used for air because that is the listing in the standard material constants table of GEANT; number 18 for the straw mixture because there are sixteen entries in the default GEANT list and 17 has been used for Mylar in a previous version of this program.

Defining the tracking medium environment

After the physical constants of the detector materials are defined, the tracking environment and tracking parameters must be specified. The routine GSTMED defines most of these; the user provides such data as the magnetic field, multiple scattering maximum, tracking precision, and a minimum step size. Here, tracking precision refers to how close a particle must be to a volume boundary before it may cross it.

The user may also change default energy cutoffs with the routine GSTPAR, standard particle parameters with GPART, and particle branching ratios and decay modes with GSDK.

In the current application, this is:

```
      CALL GSTMED
     *    (1,'SPACE$',15,0,3,20.0,5.0,0.5,0.1,0.20,0.1,0,0)
      CALL GSTMED
     *    (4,'GASTUBE$',15,1,3,20.0,5.0,0.5,0.1,0.05,0.1,0,0)
      CALL GSTMED
     *    (5,'STRAWMIX$',18,1,3,20.0,5.0,0.5,0.1,0.05,0.1,0,0)
```

This code defines three media: space, which surrounds the chamber itself; gastube, which is the material between modules and superlayers; and strawmix, which is the actual material of the tracking chamber. Both space and gastube consist of material 15, air. However: space is not a sensitive volume while gastube is; and the tracking precision is 0.20 cm in space and 0.05 cm in the gastube. Strawmix is made of the average straw material, but otherwise has the same parameters as the active gastube. These parameters, in order, are: the medium number; the medium name; material number; 1 if this is a sensitive volume, 0 else; the magnetic field tracking flag which defines the method used to track a particle within the magnetic field; the maximum magnetic field value; the maximum angle a particle is allowed to turn in one step; the maximum displacement allowed in one step due to multiple scattering; maximum energy loss allowed in one step; the tracking precision; a minimum step size; and a user-defined array of other optional parameters. It is the media numbers defined here which are used in the volume definitions described in Section 2.1.

2.3 The Physics Process Package: PHYS

An important part of the GEANT system is the PHYS package, which simulates the interaction of particles with the matter of the detector. GEANT should be accurate for processes from 10 KeV to 10 TeV, though there are some weaknesses where experimental data are weak or incomplete.

The interactions are modeled in the standard way: first, GEANT samples the total cross section to decide the probability of a given process; second, it determines the final states using the differential cross sections; and finally it computes mean energy losses, multiple scattering, delta ray production, and so on.

Hadronic interactions are calculated using GHEISHA7[3] although the use of TATINA[6] is available for backward compatibility. Electromagnetic interactions are included in GHEISHA; there are also other cross sections within GEANT itself for energies up to 100 GeV. Muon interactions are modeled up to 10 TeV. Ionization is modeled with a Landau distribution or by an explicit generation of delta rays. Similarly, a Gaussian approximation is the default for multiple scattering, though the slower and more accurate Moliere theory is also available. The interested reader should refer to the GEANT manual[2] , or, for even more detail, the GHEISHA manual[3] .

2.4 The Kinematics package: KINE

This package allows the GEANT user to store and retrieve information about event vertices and particle tracks. The routine GSVERT will store a vertex position and originating beam particle number and return a new vertex number. GSKINE stores 4-momentum, particle number, and originating vertex number for long-lived particles and returns a new track number. GFKINE retrieves the above information for a given track number. There is no retrieval routine for vertices, though the data are available from the common block GCKINE.

GPVERT and GPKINE will print the vertex and kinematic information for a given vertex or track number, respectively.

<u>The Interface to the ISAJET Monte Carlo</u>

The subroutine ISAEVEN, written by Giuseppe Ballocchi in February 1986, is the interface to the event generation Monte Carlo program ISAJET. It is not officially a part of the GEANT system, though it obviously fills an important function and is therefore included here. It reads an ISAJET output file, boosts the particles to the lab frame, and fills the GEANT track and vertex banks. It translates particle types from the ISAJET convention to the GEANT numbers. It is also the appropriate place to apply any cuts on acceptable particles to save on following those that miss the detector entirely or have such low momentum that they curl away before reaching a sensitive region of the detector. The current version applies transverse momentum and pseudorapidity cuts in this subroutine, and requires that particles be long-lived. There is also an event counter with the facility to skip the first NSKIP events in an ISAJET file before passing particles to GEANT for processing.

The following GEANT calls are in ISAEVEN, separated by reference frame translation and other code:

```
      CALL  GSVERT(VERTEX,0,0,UBUF,NBUF,NVTX)
      CALL  GSKINE(PLAB,IGNAME,NVTX,UBUF,NBUF,NTRK)
```

GSVERT stores the primary vertex, whose (x,y,z) position is stored in the array VERTEX. This vertex has originating beam track number 0, originating target track number 0, and no user buffer floating point variables (UBUF is empty and NBUF is zero). GSVERT returns NVTX, the assigned vertex number. Next, for each acceptable track, ISAEVEN calls GSKINE, which stores the particle four-momentum PLAB for particle type IGNAME from the vertex NVTX. Again there are no user parameters. The assigned track number NTRK is returned.

There are other calls to GSVERT and GSKINE from within GEANT whenever there is an interaction or decay.

2.5 THE TRACKING PACKAGE: TRAK

After a track is generated it must be propagated through the detector, all its decays and interactions with the contents of the detector calculated, and the secondaries tracked. GEANT does this by applying the equations of motion to the current particle over succesive steps and computing the four-momentum at each point. This means the particle trajectories are not perfectly smooth as in the real world; this slight inaccuracy is offset by a huge savings in computing time. GEANT has mechanisms for computing the step size, and constantly adjusts this parameter. The step size for a particle depends not only on the material through which it is travelling, but on its intrisic properties such as its lifetime, mass, and charge. The material mostly affects the step size because its radiation and interaction lengths influence particle energy loss and multiple scattering. Finally, the geometrical boundaries of the detector also come into play: the step size is limited by the path length between medium boundaries.

The GEANT user should spend some time optimising various tolerances and cuts which limit the step size. These are called the tracking medium parameters, and are stored as described in the CONS section description of the routine GSTMED and GSTPAR. They include: the maximum angle a particle is allowed to turn in one step due to the magnetic field; the maximum allowed displacement due to multiple scattering in one step; the maximum fractional energy loss allowed in one step; the tracking precision or boundary location accuracy for crossing medium boundaries; the minimum step size due to either energy loss or multiple scattering; and the energy cuts, different for each type of particle, below which a particle will not be tracked.

2.6 THE DETECTOR RESPONSE PACKAGE: HITS

There are two different kinds of detector response in the language of GEANT: hits and digitizations. A hit is detector information recorded at tracking time. It is analogous to the actual value of quantities the detector is designed to measure, such as particle position or energy loss and position. A digitization simulates the measurement of the geometrical quantity by a a detector element, such as a time and wire number or ADC signal and tower number. Presumably digitizations include all required inefficiencies and uncertainties.

The GEANT user must define both hits and digitizations, decide how the data are to be packed, and later call a GEANT utility to store them. In the current application, this means:

```
      DATA NBITSV/16,16,16/
      DATA NAMESD/'WIRE','TIME'/
      DATA NBITSD/2*16/
      DATA NAMESH/'XPOS','YPOS','ZPOS','E  '/
      DATA ORIG/500.,500.,500.,0./
      DATA FACT/4*10000./
      DATA NBITSH/4*32/

      CALL GSDET
     *    ('CENT',NAMESL,1,NAMESL,NBITSV,1,2000,500,ISET,IDET)
      CALL GSDETH('CENT',NAMESL,4,NAMESH,NBITSH,ORIG,FACT)
      CALL GSDETD('CENT',NAMESL,2,NAMESD,NBITSD)
```

The call to GSDET assigns the detector element (layer) NAMESL to be part of the user-defined detector set CENT. All layers are part of CENT: the detector set convention is an aid to storage of the detector response data. There is one volume descriptor, NAMESL. This has been defined earlier with the volume definition calls described in Section 2.1. NBITSV is a vector of dimension 3 which defines there are to be 16 bits per datum. This is user-defined detector type 1. There are 2000 words allocated at first for the primary hits bank, and

500 words allocated for the digitization banks. ISET and IDET are returned and give pointers to the set CENT and the particular detector element within CENT. GSDET is called once for every layer.

GSDETH defines the hit parameters for set CENT, subdetector NAMESL: there are 4 elements per hit, whose names are kept in the array NAMESH (x, y, z, energy); there are NBITSH (4*32) bits available for packing the variable values. ORIG and FACT define the packing of the data precisely: the ith integer variable IVAR(I) of NBITSH(I) bits is stored such that IVAR(I) = (VAR(I) + ORIG(I)) * FACT(I).

GSDETD makes a similar definition for digitizations within the set CENT, subdetector NAMESL: there are two elements per digitization, called wire and time, which are packed in 16 bits each.

The GEANT user must also store the hits and digitizations. Hits are stored in the routine GUSTEP, a user subroutine called at the end of every step. In this subroutine it is possible to decide whether to store a hit, and do so if desired. It is an easy call:

```
      CALL GSAHIT(NSET,NDET,NTRA(NT),NBV(NT),HITS(1,NT),IHIT)
```

This call stores a hit for set NSET (always 1 since there is only one detector set, CENT), subdetector NDET, track NTRA(NT), volume numbers NBV(NT) where the current step is, the array of hit elements HITS(4,NT). The hit number IHIT is returned.

When all the tracks in an event have been followed through the active detector, GEANT calls GUDIGI. In this routine the GEANT user should sort through all the hits and record digitizations. In the SSC tracking version, GUDIGI finds the wire closest to a hit, calculates the particle's distance of closest approach to this wire, and determines the signal drift time. It also removes multiple hits on a wire, taking only the first digitization if later ones are within the chamber dead time. Finally, it stores the digitizations:

```
      CALL GSDIGI(NSET,NDET,NTRA(NT),1,NBV(NT),KDIGI(1,NT),IDIG)
```

This call stores a digitization for set NSET (again, only CENT), subdetector NDET (same as module number, since there is only one superlayer per volume), track number NTRA(NT), volume numbers NBV(NT). The digitization data are kept in the array KDIGI(2,NT). The digitization number IDIG is returned.

2.7 The Drawing Package: DRAW

The drawing package is an important debugging tool. It draws the detector, making it easy to check that the geometrical specification does indeed correspond to the intended detector. It draws the detector geometry tree structure. It draws particle trajectories, making ckecks of the magnetic field and various cuts easy. Finally, it draws hits (but not digitizations), making it possible to tell at a glance if hits are being recorded properly.

Unfortunately, CERN's graphics do not interface easily to the SLAC system. SLD computer experts Dave Aston and Terry Reeves have spent immense amounts of time building a graphics interface for GEANT. Without their work most of the SSC study would have been much more difficult and much slower. Unfortunately, there are still unsolved bugs in the graphics SLAC interface. The most annonying bug loses occasional parts of the graphics output - things are sometimes just not drawn. There are theories that this is due to the GEANT-UGS interface, though the exact form has been very elusive.

The DRAW package exists in both FORTRAN subroutine and interactive forms. The two are identical: the interactive package simply translates into the the corresponding subroutine call. The difference in the calling mechanism is trivial: to draw the particle trajectory for track number 17, for example, the FORTRAN command is:

```
CALL GDXYZ(17)
```

while interactively the command is:

```
DXYZ 17
```

Notice that 'CALL G' and the parentheses are dropped in the interactive call. This is the general rule. If there are several arguments for a call, then spaces, not commas, should separate the arguments.

Because the drawing package is more efficiently used in the interactive version, all the examples that follow will be for that form; to make FORTRAN calls out of them, the GEANT programmer should generally add 'CALL G' at the beginning, commas between the arguments, and parentheses around the argument list.

Drawing the Detector

There are several ways to draw cuts and projections through the detector. A very useful one is DCUT, which has as arguments: the detector name; the axis which is normal to the cut plane of the view (that is, the axis along the line-of-sight); the distance from the origin the cut plane is placed; the u and v coordinates of the volume origin on the screen; and the u and v scale factors. Note that DRAW calculates the detector origin placement with the scale factors: the picture placement will change if the origin stays the same but the scale factor changes. The visible part of the screen is a 20.×20. square, with the lower left corner at (0.,0.) and the upper right corner at (20.,20.). This means the center is at (10.,10.), not (0.,0.) as the unwary user might suspect.

To draw the entire detector GLOB sighting along the z-axis (the 3-axis) but otherwise centered requires a scale factor of 0.05. The interactive command for this is:

```
DCUT GLOB 3 0 10.  10.  .05 .05
```

Notice the axis definition is an integer while the other numerical arguments are real numbers. In general GEANT requires the proper form for its arguments, though it understands the '0' which places the cut plane at the detector origin properly. Both upper and lower case work. Figure 2a shows the output of this command.

Naturally, this important call does not follow the general naming convention.

The FORTRAN is 'CALL GDRAWC(parms)'.

Invoking any of the detector drawing commands sets position and scale parameters which all subsequent drawings will use until the next detector command. These variables are not set at execute time, and must be set initially by DCUT or another detector drawing command. The impatient user can draw only one module (superlayer) to save time:

```
DCUT MD01 3 0 10.  10.  .05 .05
```

The commands:

```
DCUT MD12 3 0 10.  -140.  1.  1.
DCUT MD13 3 0 10.  -140.  1.  1.
```

move the detector center to (10.,-140.) and the scale factor is set to 1.: figure 3a shows the result. The layer lines are polygonal, not circular as defined, a vestige of how the graphics are set up. Internally, the layer boundaries and positions are correct.

Any scale factor and any detector center position is possible.

Clearing the Screen

The command:

```
NEXT
```

will clear the screen for the next drawing.

Drawing Tracks

Particle trajectories may be superimposed on a drawing of the detector or not. Particle numbers and names may be added, although they are drawn only where a particle decays or leaves the detector.

```
DXYZ 17
DPART 17
```

The first line will draw the trajectory of particle 17, the second will draw the track number and name of particle 17. No particle specification or particle number 0 will process all the particles. Muon tracks produce dashed lines in the current version of DRAW; all others produce solid lines. Figure 2b shows all the tracks and particle identifications for a minimum bias 40 TeV SSC event. Figure 3b shows a portion of the same event expanded to life size, created after the coordinate system was redefined with a DCUT command. Both 2b and 3b were created using:

```
DXYZ
DPART
```

<u>Drawing Hits</u>

```
DHITS 17
DHITS
```

The first line will draw the hits for particle 17; the second for all particles. Figures 2c and 3c show the hits for all particles for the whole detector and a portion of the detector lifesize, respectively. Notice that the crosses which designate the hits do not always intersect the lines which represent the detector layers. This is because the layers are incorrectly drawn as a polygon, as discussed earlier.

2.8 THE I/O SERVICE PACKAGE: IOPA

The I/O package permits the user to read or write selected data structures to and from external media. Since these utilities use ZEBRA packages, it is possible to write to tape in a machine-independent format. This facilitates the use of several machines for one GEANT project.

<u>Opening and Closing a Logical Unit</u>

Each unit must be opened before GEANT can read from or write to it. All units should be closed before the end of the program. This is easily done with calls to GOPEN and GCLOSE. In the current example, this is:

```
CALL GOPEN (LUN, 'I', LEN, IERR)
CALL GLOSE(LUN, IERR)
```

These open and close unit LUN for Input (also available are Output and Exchange), with a maximum record length of LEN (set to 80). An error flag IERR is returned in each case.

Reading and Writing Data Structures

GEANT data are classed loosely into two types. Initialization data are general throughout the run and include the particle parameters, the materials data, the media data, the volume parameters, rotation matrices (not discussed here), the detector set which includes hit and digitization parameters, and drawing parameters. A reference to the structure INIT means all the above structures. Event-wise data include the vertex and kinematic data; the trajectory space points; the hits; and the digitizations.

The GEANT user should record or read the initialisation data at the beginning of each run, then the desired event-wise data once per event. This is easily done with a call to GSAVE or GGET. These have as an argument list the desired data structures and an opportunity to flag the routine that only the initialisation routines from the list should be processed.

In the current SSC case, this is:

```
CALL GSAVE(60,LSAVE,-NSAVE,IDENT,IER)
CALL GGET(60,LGET,-NGET,IDENT,IER)
```

Both calls have the same arguments: the unit number to read/write, a list of data structures to process, the number of structures in the list, a returned record identifier, and a returned error. If the number of structures to process is negative as in the above example, then GEANT picks out only the initialisation data structures. If it is positive, it processes only the event-wise structures.

Reading From Multiple Volumes

A common application is the writing of GEANT data onto several tapes and then reading from them in order to run an analysis. ZEBRA, unfortunately, does not allow for multiple volumes on a file number. The usual FORTRAN volume number incrementing does not work because ZEBRA's tape handling facility rewinds the tape and resets the volume number when it comes to the end of data. There seems to be no way of getting around this; the multivolume user must name each tape a new unit and reset the unit number within the GEANT program. One solution is to run the following code whenever ZEBRA comes to an end-of-data mark:

```
NSEQH = NSEQH + 1
IF(NSEQH .LE. MAXSQH) THEN
   WRITE(LOUT,'('' About to close unit'',I4)')LUNHGS
   CALL FZENDI(LUNHGS,'T')
   LUNHGS = LUNHGS + 1
   WRITE(LOUT,'('' About to open file number'',I4)')NSEQH
   WRITE(LOUT,'('' Will now start reading from unit'',I4)')
*        LUNHGS
   ISTAT = 0
   CALL FMOUNT(LUNHGS,IONE,ISTAT)
   IF(ISTAT .NE. IONE)IEND = 1
   WRITE(LOUT,'('' Completed FMOUNT with ISTAT'',I4,
*        '', IEND'',I3)') ISTAT, IEND
   CALL FZFILE(LUNHGS,LEN,'I')
ELSE
   IEND = 1
   WRITE(LOUT,'('' End of data on unit'',I3)')LUNHGS
ENDIF
RETURN
```

This bit of code increments a sequence number and checks that it is within a range set earlier. It then closes the current input file with FZENDI and increments the unit number. It asks for the mounting of the next tape with FMOUNT, which sends a message to the computer operator to hang the tape assigned to unit number LUNHGS. Then it opens the file LUNHGS with a call to FZFILE, and continues.

Because defining several tapes to be mounted sequentially to one tape drive can be tricky at SLAC, here are the commands which will work. They belong in the job's EXEC file, or wherever FILEDEFs are usually placed.

```
'SETUP TAPE 181 SU4544 SL NORING '
'SETUP TAPE 181 SU4543 SL NORING '
'SETUP TAPE 181 SU4543 SL NORING '
'SETUP TAPE 182 SU4445 SL RING '
'SETUP END'
dcb=' (RECFM VBS LRECL 19996 BLKSIZE 20000'
'FILEDEF FT60F001 TAP1 SL 2 VOLID SU4544' dcb
'LABELDEF FT60F001 VOLID SU4544'
'FILEDEF FT61F001 TAP1 SL 3 VOLID SU4543' dcb
'LABELDEF FT61F001 VOLID SU4543'
'FILEDEF FT62F001 TAP1 SL 4 VOLID SU4543' dcb
'LABELDEF FT62F001 VOLID SU4543'
'FILEDEF 80 TAP2 SL 1 VOLID SU4445' dcb
```

These commands ask for two tape drives, called 181 and 182. Each tape or file on a tape is assigned separately with the SETUP command. The FILEDEFs define unit 60 to be on the unit TAP1 (which is the same unit as TAPE 181), a standard label tape, file 2, volume ID as given. FMOUNT requires a LABELDEF, which is simply a check on the attached tape identifier. Unit 61 is defined to be the next tape, and unit 62 corresponds to the next file on the same tape. Unit 80 is an output tape, and is designated to be mounted on the unit TAP2, or TAPE 182.

It is not possible to write to more than one tape in one job.

2.9 The Interactive Package: XINT

The interactive version of GEANT is an important tool for designers of detectors and debuggers of programs. The user may call any of the basic functions of GEANT, in effect designing a detector interactively. This means it is possible to design or modify the detector geometry, change the media parameters, and manipulate the running conditions on an event-by-event basis. As described in detail in Section 2.7, it is possible to draw the detector, the particle trajectories, and the hits. It is also possible to debug programs more quickly and easily using interactive GEANT.

The package is based on the ZCEDEX[7] command processor, though a minimal knowledge of ZCEDEX is required. In general the interactive commands parrot the regular FORTRAN calls exactly, with minor name changes and extremely rare argument changes. The GEANT user who understands the FORTRAN version should have no problems using the interactive form.

2.10 The Job Flow

Generally a GEANT program has an initialization section followed by a loop over an event-wise stepping through the detector. The initialization section defines and initializes space allocation for HBOOK and ZEBRA, initializes the GEANT physics data and drawing package, loads in the particle data, and defines the detector geometry.

The event-wise loop generates or inputs the event kinematics (in the SSC example, from ISAJET), stores the event vertex, and then starts looping over tracks. The track-wise loop includes all secondaries, which are added to the track list as they are created. GEANT follows the track step by step, checking to see if the particle has entered a new volume or interacted. If the particle has entered or left an active volume, GEANT calls GUSTEP, thus giving momentary control to the user. If the particle has interacted, it stores new tracks and vertices.

After all the primary and secondary tracks are processed, GEANT calls GUDIGI, again giving control to the user for digitization purposes. The next call is to GUOUT, where the user may decide whether to store an event and may increment counters. This is the last call in the event-wise loop.

After the desired events have been processed (or the job is close to its time limit), GEANT closes its files and outputs histograms.

A flow chart of the SSC tracking program is in figure 4.

3. Running GEANT at SLAC

It is not difficult to run GEANT at SLAC, though some care must be taken to build the correct machine environment. This chapter describes how to construct the vitrual machine environment for either interactive or batch GEANT and gives names and locations of working examples.

3.1 SETTING UP THE VIRTUAL MACHINE

The GEANT program with interactive graphics requires nearly 4M virtual memory to run; this does not include memory needed by ZEBRA to manipulate and store the resulting data. In practice, an 8M machine is necessary to run a useful GEANT program. Even with such a large machine environment, it is not possible to run GEANT with a debugger.

The command

```
q storage
```

shows how much virtual memory the current machine has.

```
dirmaint storage 8m
```

invokes the Directory Maintenance Program to change the virtual machine size to 8M, and is performed only once. The machine size will not change until another dirmaint storage command is executed. Users need to have special dispensation to have access to an 8M machine.

The GEANT user must link to a few disks in order to have access to various TXTLIBs and other files. It is also very useful to have about 10 cylinders of space disk available to store GEANT output during program development. The following EXEC file asks for a 10 cylinder space disk, puts it into the user's A-disk slot (leaving the user's 191 disk in the B-disk spot), and attaches other disks required for GEANT.

```
/* SET UP ENVIRONMENT FOR RUNNING GEANT */
Trace Off
'SET CMSTYPE HT'
 "CP .LINK * 330 330"
 IF RC /= 0 THEN DO
  "SPACE ADDTEMP 330 10"
  COUNTER = 1
  DO UNTIL RC=0 | COUNTER=10
    "CP SLEEP 3 SEC"
    "CP LINK * 330 330"
    COUNTER = COUNTER + 1
  END
 END
 "SWAP A B"
 "ACCESS 330 A"
'SET CMSTYPE RT'
 "GIME PUBEH 501 C ( QUIET"  /* the main GEANT disk */
 "GIME PUBEB 1A4 D"  /* has GEANT311 and other code */
 "GIME PUBEB 198 H"  /* has user-supported code, examples */
 "GIME UGS77 Q"  /* the Unified Graphics disk*/
 "NEWS NEW (C"
Exit RC
```

A slightly different version of this EXEC can be found in GEANT EXEC on APP 191.

3.2 Other Necessary Files

The user must also have a copy of the GEANT control cards available for the program. These control the running of the job, with specifications for the number of events to run, physics processes to include, debug and other switches, and data structures to save or read. An example of these cards is in GEXAM8 GEANTDAT on the APP 191 and is listed below. Other examples on PUBEH 501 and the PUBEB 198 disk are all called GEXAMn GEANTDAT, with n an integer from 1 to 8.

```
LIST
TRIG 5     (process 5 events)
DEBU 1 6 1     (debug from 1st to 6th event, by ones)
SWIT 1=1 2=0 3=0 4=0 5=0     (user debug flags)
ANNI 1     (annihilation flag)
BREM 1     (bremsstrahlung flag)
COMP 1     (Compton scattering flag)
DRAY 1     (delta ray flag
HADR 1     (hadronic process flag)
LOSS 1     (energy loss flag)
MULS 1     (multiple scattering flag)
MUNU 1     (muon nuclear interaction flag)
PAIR 1     (pair production flag)
PHOT 1     (photoelectric effect flag)
SAVE 'INIT' 'DIGI' 'KINE' 'JXYZ' 'HITS'     (save these ds)
PRINT 'MATE' 'VOLU' 'TMED'     (print these data structures)
END
```

A complete summary of the GEANT data cards is in the manual at BASE040-2. Meanings of the physics flags are at PHYS001-3.

There are several examples of user code on the GEANT and APP disks; these are called GEXAMn FORTRAN. The system requires that the integer n in the

FORTRAN and GEANTDAT file names match. The current version of the SSC tracking GEANT program can be found in GEXAM8 FORTRAN on APP 191; a more stable but less current version is in GEXAM9 FORTRAN on the same disk.

The GEANT source code is in GEANT311 FORTRAN and GEANG311 FORTRAN on PUBEB 198.

The SSC version also requires an ISAJET output file of events which are to be processed by GEANT. Naturally, each user will want to have the appropriate events to put through the detector simulation; however, for preliminary playing a file of quark jet events called ISAJET DATA is available on BON 191. It should be placed on the space disk at A.

3.3 Running GEANT Interactively

Once the virtual machine is set up and all needed files are in place, it is easy to run a GEANT simulation interactively: simply command

```
geantint gexamn
```

This invokes the EXEC file GEANTINT, which sets up the required file definitions, load libraries, and so on. It then invokes GOGEANT EXEC, which finishes setting up the running environment. It then loads, links, and executes GEANT with the user programs in GEXAMn FORTRAN and program control cards in GEXAMn GEANTDAT. If GEXAMn has not been compiled, GEANTINT will do it if the command is:

```
geantint /ft gexamn
```

Here, the /ft parameter causes GEANTINT to compile GEXAMn before linking.

After GEANT is linked and loaded (which may take several minutes) and the geometry defined, the command

```
trig
```

will process the first event. This may take quite some time; the SSC version has trackwise data printed to the screen so the user may follow the program's progress. After the event is processed, GEANT will output the message:

```
Type EXIT or <RETURN> thrice to get out
```

and the user may command TRIG again to do the next event, or enter other interactive commands as described in Sections 2.7 and 2.9 or in the GEANT manual. It is important to clear the screen once before beginning to draw to it; otherwise the first drawing may not be scaled properly. This is done with:

```
next
```

The command

```
exit
```

will finish an interactive session.

The execute files described here open and close a console file for the user. This console file has a name `username CON`nnnn, with a form of the date for nnnn, and is sent to the user's reader. It can be treated as any other reader file.

The current version of GEANT outputs all graphics to the file GOGEANT SEQ4010. There is no way to print only part of the graphics output generated during a given session; although Dave Aston has a solution, it has not yet been installed into the SSC version. The command

```
tekprint gogeant seq4010 (pref imcgb1
```

prints the graphics file in preformatted mode on the printer on the first floor of the computer building, IMCGB1. It is important to use the preformatted form of the tekprint command for these complicated drawings.

3.4 Running GEANT in Batch

To run GEANT in batch simply submit the EXEC file GEANTBAT. This sets up the BATCH environment, then executes GOGEANT. Examples of GEANTBAT are on APP 191 and the GEANT disk PUBEB 501. It is important, however, to remember that the batch machine requires the files it executes to be on the submitting user's A-disk or on another disk explicitly attached by the job.

To submit a batch job with an already compiled user program in GEXAMn TXTLIB, enter:

```
batch submit (tim 8 tapes) geantbat gexamn
```

This submits an 8-minute job with tape setups. The other defaults such as an 8M machine required for running are already stipulated inside the GEANTBAT and GOGEANT EXECs on APP 191.

GEANTBAT also has an /ft parameter:

```
batch submit (tim 4 notapes print 80k) geantbat /ft gexamn
```

will compile GEXAMn FORTRAN before linking.

The available versions of GEANTBAT make use of the BATCH logging facility: the EXEC prompts the user for comments about this particular job and keeps the results in a file called GEANT BATCHLOG on the user's 191 disk. It also assigns sequential names GEANTnnn, where nnn is a job number incremented only with GEANT jobs submitted this way. This is very useful for keeping GEANT jobs separate from other tasks and for keeping track of programming progress.

4. The ZEBRA Data Management System

GEANT now uses the data manager ZEBRA. It was originally written using the data manager ZBOOK and the use of ZEBRA is largely oriented along ZBOOK rules; the implementation of ZEBRA in GEANT is not optimal. Nevertheless, although in theory the GEANT user need know nothing about memory management, in practice some knowledge of ZEBRA is very useful. This section is a short introduction to the basic ideas of ZEBRA.

One major defect of FORTRAN is its lack of data structuring facilities. Its only 'structures' are arrays and common blocks, both of which must be defined by compile time. Since FORTRAN stores only locations of the beginning of arrays and commons, neither of these may be manipulated as an entity. The FORTRAN user spends much programming time manipulating data storage, or (more likely) wastes much storage space keeping empty or partially empty structures.

ZEBRA is an attempt to offer true dynamical data management. Written at CERN, it is a collection of FORTRAN77 routines which provides a sophisticated system of data management. The routines are grouped into three packages, called MZ, FZ, and DZ or DIA. The memory manipulation package MZ contains all the initialization, allocation, and bank manipulation routines. The file management package FZ contains all the I/O routines. The diagnostics package DZ contains methods of displaying and verifying data structures. In general a routine within a package will have a name that begins with the appropriate two letters.

The ZEBRA pointers and links are available within COMMON blocks, making the data available to the user. It is also possible to arrange the data storage in such a way as to keep relational information intact.

4.1 Data Structures

The ZEBRA Bank

The basic unit of storage within ZEBRA is the bank. This is a contiguous area of storage which includes I/O descriptor words, reference and structural links as pointers to data within the bank, addresses of the supporting and next banks, bank status words, and data. A program may have one bank or many.

The Data structure

A data structure is a collection of banks which are associated with one another in some way. This association is determined by how they are linked together. Because banks are created dynamically at execution time, and because each bank contains its own structural links, there may be an arbitrary number of banks in a given application. This means there is no need to define a maximum dimension as for arrays or common blocks. The number of banks is limited only by the amount of storage allocated for ZEBRA itself.

The simplest data structure is the linear structure. In this arrangement, each bank has a link called the 'next link' which points to the next bank in the system. A next link of zero means there are no more banks in the linear structure. Figure 5 shows a representation of a simple linear structure.

To access a linear structure, it is sufficient to point to its first bank. Then all the other banks in the structure are available.

A more complicated structure is needed in many applications; for that reason there are also 'down links' and 'up links.' A down link points to a bank which somehow depends on the originating bank. For example, a vertex bank may have down links to all the track banks for particles which originate at that vertex. Or a detector volume bank may have a down link to the first of all the hits in that detector volume. A bank may have a large number of down links. A down link may point to the first bank of a linear structure.

An up link is the reverse of a down link: each bank has one up link which points to the bank on which it or its linear structure depends. If the up link is zero, then a bank is at the top of its tree.

There are also origin links, which point to the structural link that supports the bank. That is, it points to the down link of the previous bank.

Figure 6 shows a representation of the hits data structure JHITS in GEANT, which has all four kinds of links.

Reference Links

The use of the four structural links described above defines the form of the data structure: an intelligent structure design keeps the data ordered in an intuitive format which is easy to understand and remember. If a user wishes to establish links between banks that do not define the structure itself, reference links are available. These links keep references that the user wishes to record, but do not affect the data structure itself in any way. ZEBRA's actions on reference links are limited to reassigning them if banks are moved within memory.

4.2 Physical Storage

ZEBRA's banks are kept in one or more contiguous areas of storage whose number and sizes are defined by the user at initialization time. These are called dynamic stores; each one resides in a separate common block. ZEBRA can handle up to 16 dynamic stores, though there are computer time overheads associated with the number of times ZEBRA must do something in a store other than the 'current' one. GEANT has only one very large store, which in the SSC application is:

```
PARAMETER ( MZEBR$ = 1000000 )
COMMON/GCBANK/Q(MZEBR$)
DIMENSION IQ(1),Q(1),LQ(8000)
EQUIVALENCE (Q(1),IQ(1),LQ(9)),(LQ(1),LMAIN)
```

and is initialized with a call to MZSTOR. Here one million words (!) are allocated to the dynamic store kept in the common block GCBANK. The effect of the EQUIVALENCE statement is to offset the arrays Q and LQ by eight locations. This is because there are eight structural links and other identifiers in a bank between the next link position and the beginning of the bank's data. Then links are referred to as LQ(L+n) while data are IQ(L+n) or Q(L+n), where L is the offset due to the bank's location within the store. Figure 7 shows the format of a ZEBRA bank.

Divisions

Each dynamic store has three divisions by default, although a different number may be created using MZDIV. The first division is used by the system; the other two are available to the user. Divisions associate banks which are somehow logically connected, and make I/O and bank dropping of these associated banks easier and faster. ZEBRA is also more efficient at handling links within a division. The dynamic store in GEANT has two divisions IXCONS and IXDIV, corresponding to initialisation (constant) data and event-wise data.

Link Areas

A user who wishes easy access to bank links should define a common block called a link area. ZEBRA will then maintain the links. GEANT has a link area which contains all the links to the several data structures:

```
      COMMON/GCLINK/JDIGI, JDRAW, JHEAD, JHITS, JKINE, JMATE,
     + JPART, JROTM, JRUNG, JSET, JSTAK, JGSTAT, JTMED, JTRACK,
     + JVERTX, JVOLUM, JXYZ
```

Working Space

Short-term working space is available within ZEBRA at the first part of a dynamic store. This is created by a call to MZWORK, and in GEANT exists in 9000 words near the beginning of GCBANK.

4.3 Maintenance of the Dynamic Store

When a program using ZEBRA begins, the dynamic store contains only a few system banks. As other banks are created, available space decreases. After banks are no longer needed they may be dropped with a call to MZDROP. After a bank is dropped the data stay intact until a bank reorganization takes place. When there is insufficient empty storage to create a new bank, ZEBRA will perform a garbage collection. In this procedure, which is undertaken automatically when there is insufficient memory to perform a requested operation, ZEBRA moves active banks to one contiguous area near the beginning of the dynamic store. This removes and overwrites old data from dropped banks. ZEBRA naturally resets all the links to point to the banks' new locations.

If a request for memory cannot be satisfied even after garbage collection ZEBRA experiences a fatal error and the job exits to QNEXT. In the current implementation of GEANT, this causes the job to die. Since any event simulation occasionally produces an exceptionally large event, an important improvement would be to have ZEBRA simply terminate the current event and continue.

Another way of freeing memory space for immediate use is by wiping out entire divisions. This is done by a call to MZWIPE. It is particularly useful in GEANT: when all the event-wise data are no longer needed, a single call will erase them and make large amounts of space available for processing the next event.

4.4 I/O

Writing to and reading from external media is quite easy within the ZEBRA system. The user need only call the appropriate input/output routine; ZEBRA maintains the data structures and their links. All of ZEBRA's file manipulation routines are in the FZ package. The FZ package writes data either in 'native' or 'export' mode. 'Native' mode data are written in the representation of the machine at which the program is currently running, while 'export' mode data are written such that they can be read by most of the computers used by the HEP community. There is also an RZ subpackage which is a database management system with sequential and direct access to data. It is a simple system but quite adequate for the needs of HEP experiments.

GEANT uses the I/O routines FZIN and FZOUT, although calls to these are in the GEANT routines GGET and GSAVE and are not normally visible to the GEANT user. However, as discussed in Section 2.8, reading from multiple volumes in the ZEBRA system is difficult and some tricks are necessary.

4.5 DEBUGGING

In the current incarnation of ZEBRA within GEANT, the program alone requires an 8 Mbyte machine and is simply too large to run with a debugger. However, there is a diagnostic package DZ which includes methods for displaying or checking part or all of a dynamic store.

The user communication array IQUEST also includes information for debugging. Kept in the common block QUEST, this array contains many pointers and error flags which describe the problem. A detailed explanation of these is in the User Reference Guide Book DIA.

Acknowledgements

I am indebted to Gail Hanson for many exciting and fruitful discussions as well as a careful reading of the text. Thanks are also due Dave Aston for help in understanding GEANT and to Traudl Hansl-Kozanecki for help in understanding ZEBRA.

FIGURE CAPTIONS

1. The geometry tree structure of the SSC central tracking detector.

2. a: The entire SSC tracking detector. The innermost single and outermost double lines are for the volumes GLOB and CHAM. The thirteen modules of eight layers each are clearly visible.

   b: A 40 TeV minimum bias event in the SSC central tracker. Particle names and numbers are also drawn.

   c: Hits from a minimum bias event in the SSC central tracker.

3. a: The top portion of the SSC central tracker, life size.

   b: A minimum bias event in the SSC central tracker, life size.

   c: Hits from a minimum bias event in the top portion of the SSC central tracker, life size.

4. A flow diagram of the SSC tracking GEANT program.

5. A representation of a simple linear structure, from Reference 5.

6. A representation of the Hit data structure JHITS in GEANT.

7. The format of a ZEBRA bank, from Reference 5.

REFERENCES

1. G.G.Hanson, et al., Proceedings of the Workshop on Experiments, Detectors, and Experimental Areas for the Supercollider, (1987) p. 340

2. R. Brun, F. Bruyant, and P. Zanarini, GEANT3 Users Guide, CERN DD/EE/84-1 (February 1985, rev. September 1987)

3. H. Fesefeldt, The Simulation of Hadronic Showers, PITHA 85/02

4. R.L.Ford and W.R.Nelson, SLAC Report 210 (1978)

5. R. Brun and J. Zoll, ZEBRA User Guide, CERN Computer Library Long Write-Up Q100, PITHA 85/02, (January 1987)
   J. Zoll, ZEBRA Reference Manual, Book DIA (January 1986)
   J. Zoll, ZEBRA Reference Manual, Book FZ (July 1987)
   J. Zoll, ZEBRA Reference Manual, Book MZ (January 1986)

6. T. Baroncelli, INFN - Roma

7. ZCEDEX User Guide, CERN DD/EE/80-6

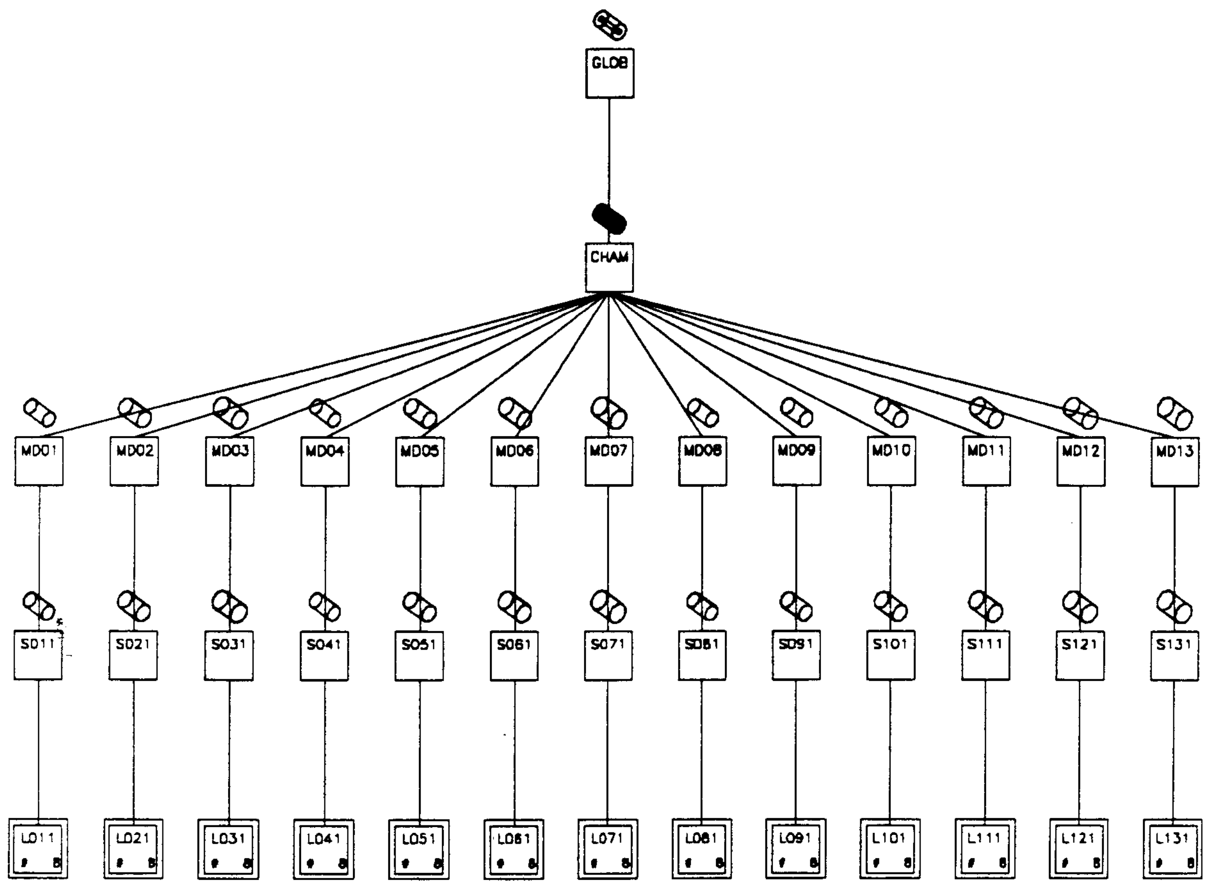

Figure 1: The SSC central tracking detector geometry tree structure.

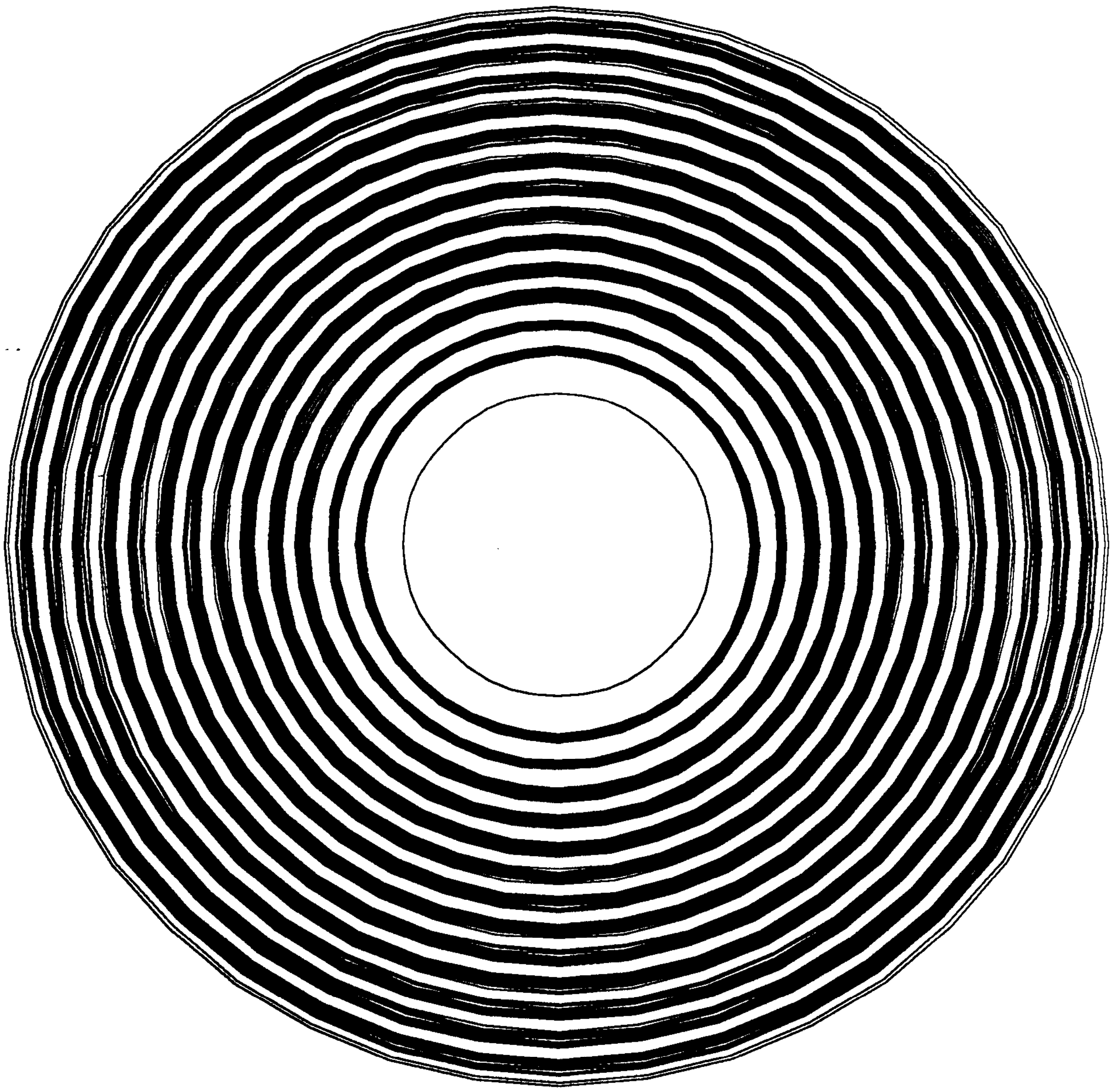

Figure 2a: The entire SSC tracking detector. The innermost single and outermost double lines are for the volume GLOB. The thirteen modules of eight layers each are clearly visible.

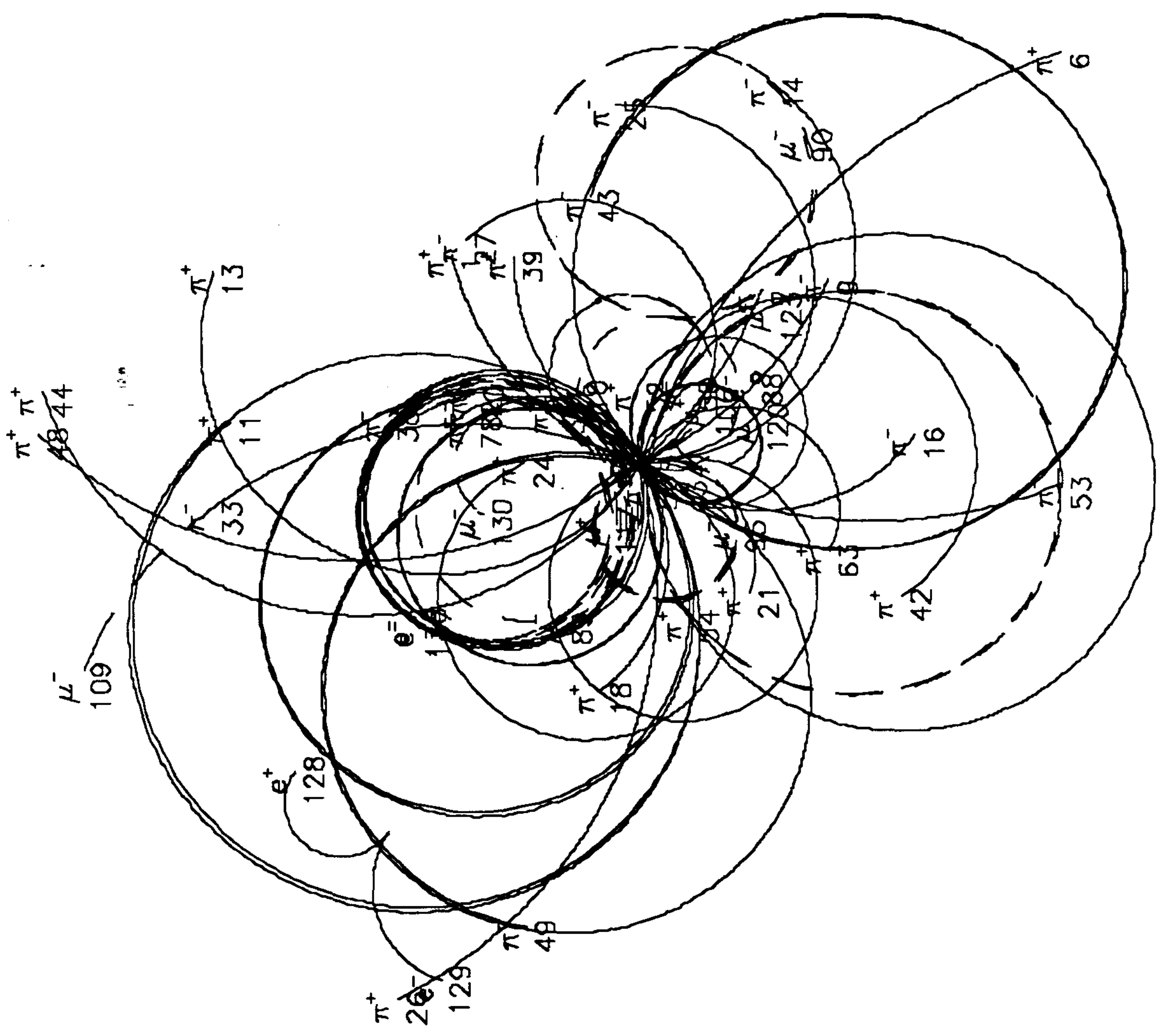

Figure 2b: A 40 TeV minimum bias event in the SSC central tracker. Particle names and track numbers are also drawn.

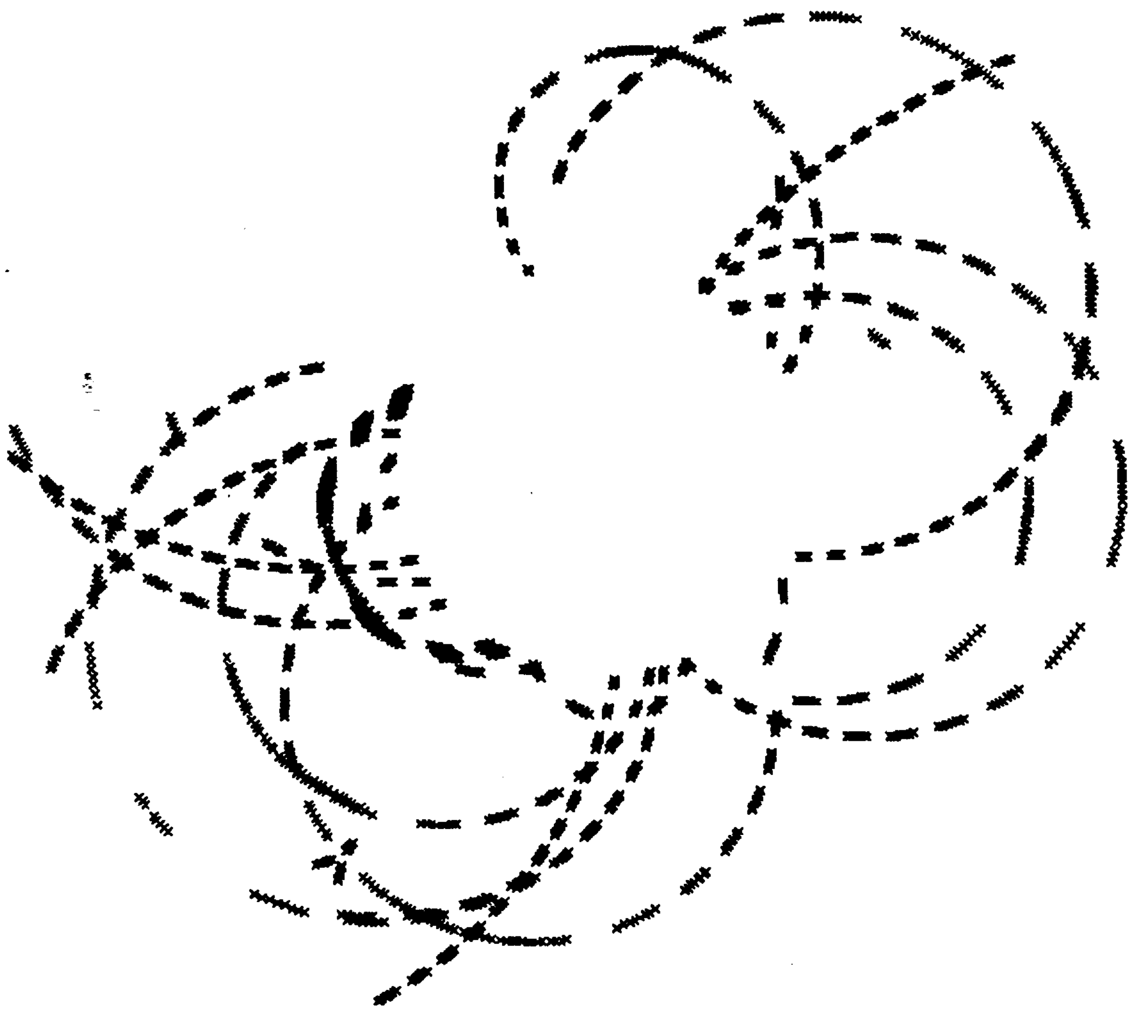

Figure 2c: Hits from a minimum bias event in the SSC central tracker.

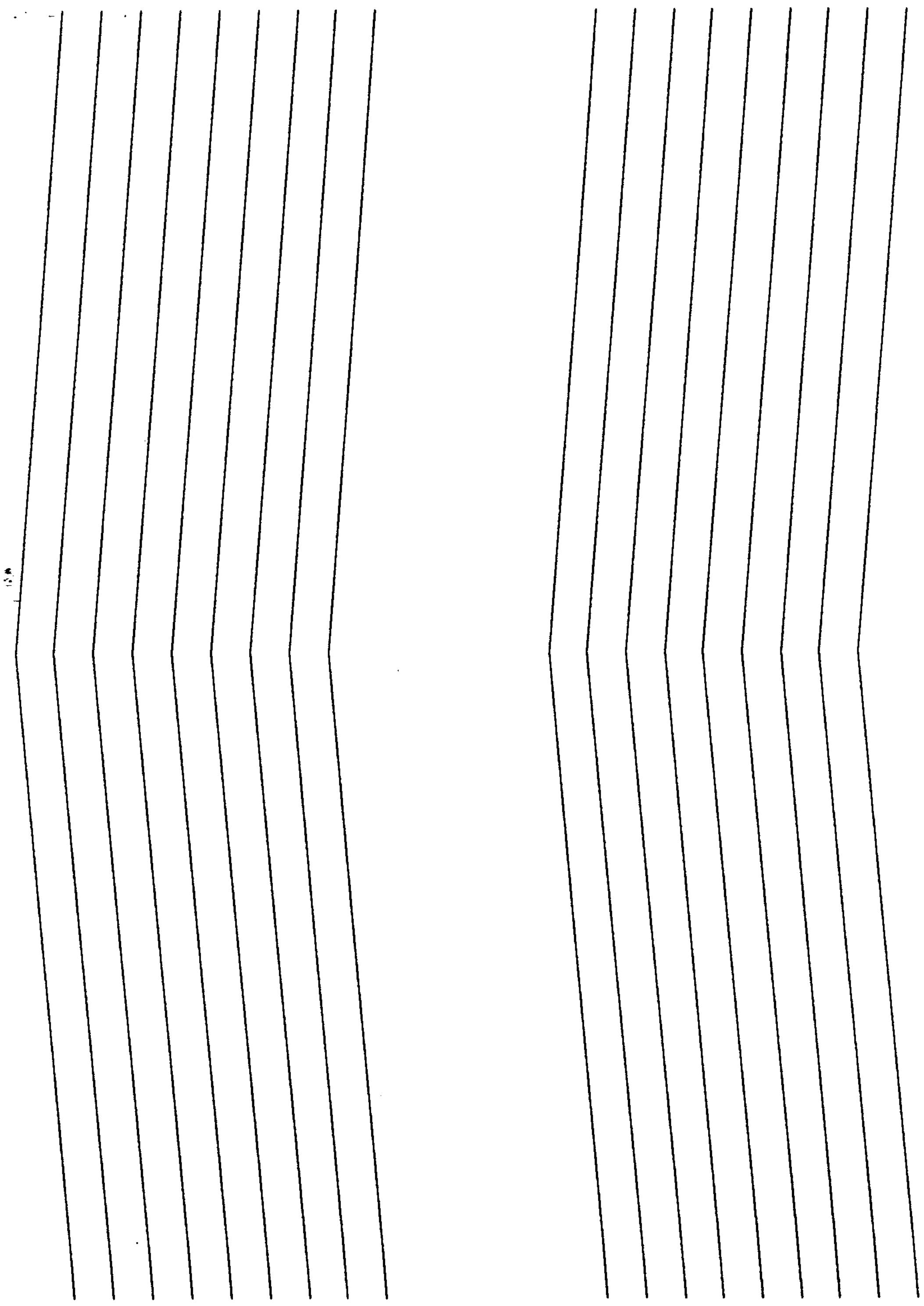

Figure 3a: The top portion of the SSC central tracker, life size.

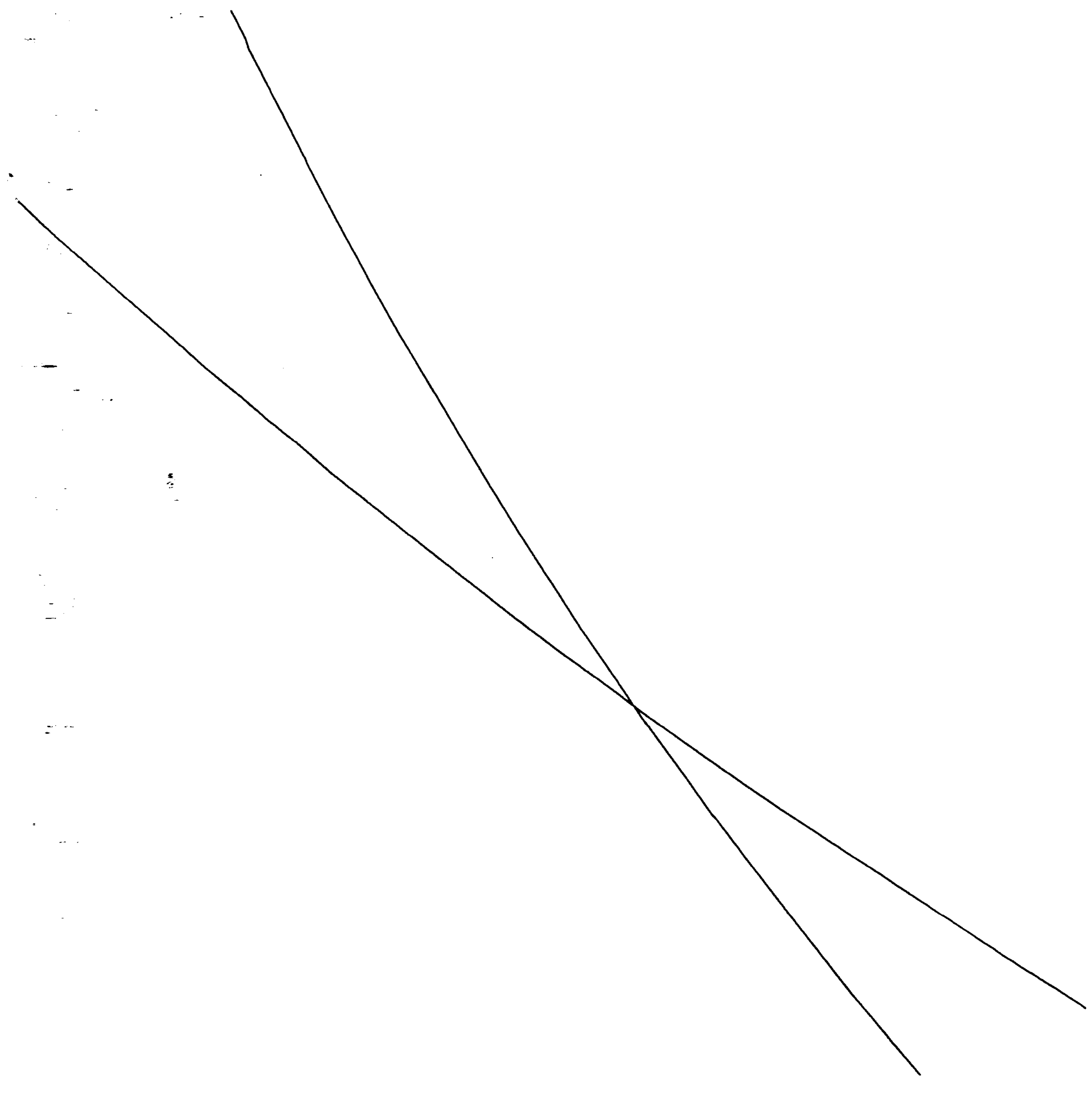

Figure 3b: A minimum bias in the SSC central tracker, life size.

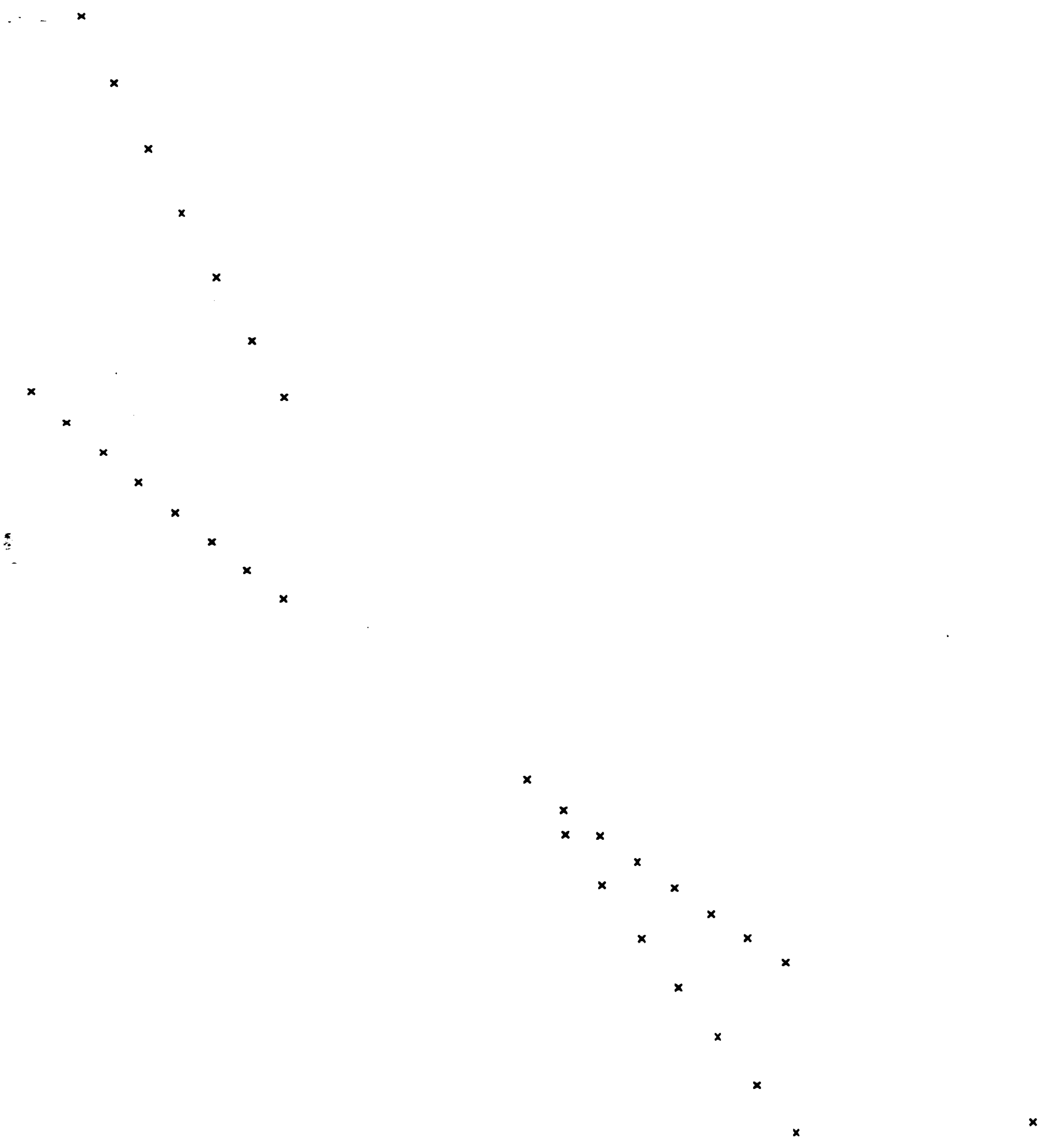

Figure 3c: Hits from a minimum bias event in the top portion of the SSC central tracker, life size.

```
MAIN
|--GUINIT  initialise GEANT and USER program and data cards
|    |--GZEBRA   initialise ZEBRA
|    |--HLIMIT   initialise HBOOK
|    |--GINIT    initialise GEANT3
|    |--GZINIT   initialise GEANT3
|    |--GDINIT   initialise GEANT drawing package
|    |--GPART    load in particle data book
|    |--GEOCAL   detector geometry
|    |    |--GSMATE
|    |    |--GSTMED
|    |    |--GSVOLU
|    |    |--GSPOS
|    |    |--GSDVN
|    |    |--GSDET
|    |    |--GSDETH
|    |    |--GSDETD
|    |    |--GSORD
|    |    |--GGCLOS
|    |
|    |--GLOOK/GPRINT   print detector definitions
|    |--GPHYSI   physics for showering
|    |--UHINIT   user hist definitions
|    |    |--HBOOK1
|    |    |--HBIGBI
|    |
|    |--GSAVE/GGET
|
|--GRUN   loop over events
|    |--GTRIGI initialisation for event processing
|    |--GTRIG  process one event trigger
|    |    |--GUKINE   generate or input event kinematics (from ISAJET)
|    |    |    |--ISAEVEN
|    |    |    |--GSVERT   store primary vertex
|    |    |    |--GSKINE
|    |    |
|    |    |--GUTREV   loop over tracks, including secondaries
|    |    |    |--GTREVE
|    |    |         |--GUTRAK
|    |    |              |--GTRACK
|    |    |                   |--GTSET   initialise physics processes
|    |    |                   |--GMEDIA
|    |    |                   |--GFINDS  fill /GCSETS/ according to /GCTRAK/
|    |    |                   |--GUSTEP  when entering and leaving volume
|    |    |                   |--GTVOL   when inside volume: extrapolate track to exit point of current volume
|    |    |                        |--GNEXT
|    |    |                        |--GT.... track particle, by type (GTGAMA, GTNEUT, GTHADR, GTMUON, GTINO)
|    |    |                        |--GFSTAT   fill bank for volume statistics
|    |    |                        |--GUSTEP   after each step
|    |    |                        |    |--GSXYZ
|    |    |                        |    |--GOKING
|    |    |                        |    |    |--GSVERT  store vert
|    |    |                        |    |    |--(GSSTACK)
|    |    |                        |    |    |--GSKINE
|    |    |                        |    |    |--GSVERT (2o)
|    |    |                        |    |    |--GSKINE (2o)
|    |    |                        |    |
|    |    |                        |    |--UHTOC  determine volume
|    |    |                        |    |--GICYL
|    |    |                        |    |--GSAHIT
|    |    |                        |
|    |    |                        |--GMENEW   into new volume
|    |    |                        |--GUSTEP   if still in same volume, start over with call to GNEXT
|    |    |
|    |    |--GUDIGI
|    |    |    |--GFHITS
|    |    |    |--GSAHIT
|    |    |    |--GSDIGI
|    |    |
|    |    |--GUOUT
|    |         |--GSAVE/GGET
|    |         |--GPRINT   according to switches
|    |
|    |--GTRIGC
|
|--UGLAST
|    |--GLAST
|    |--HISTDO
|
stop
```

Figure 4: A flow diagram of the SSC tracking GEANT program.

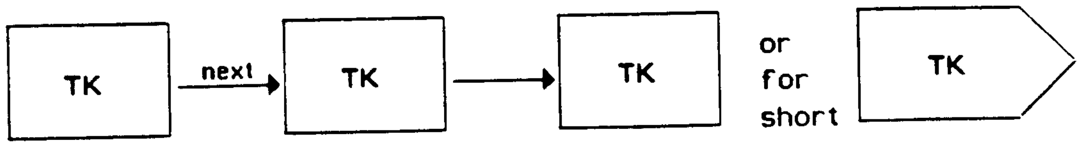

Figure 5: A representation of a simple linear structure, from Reference 5.

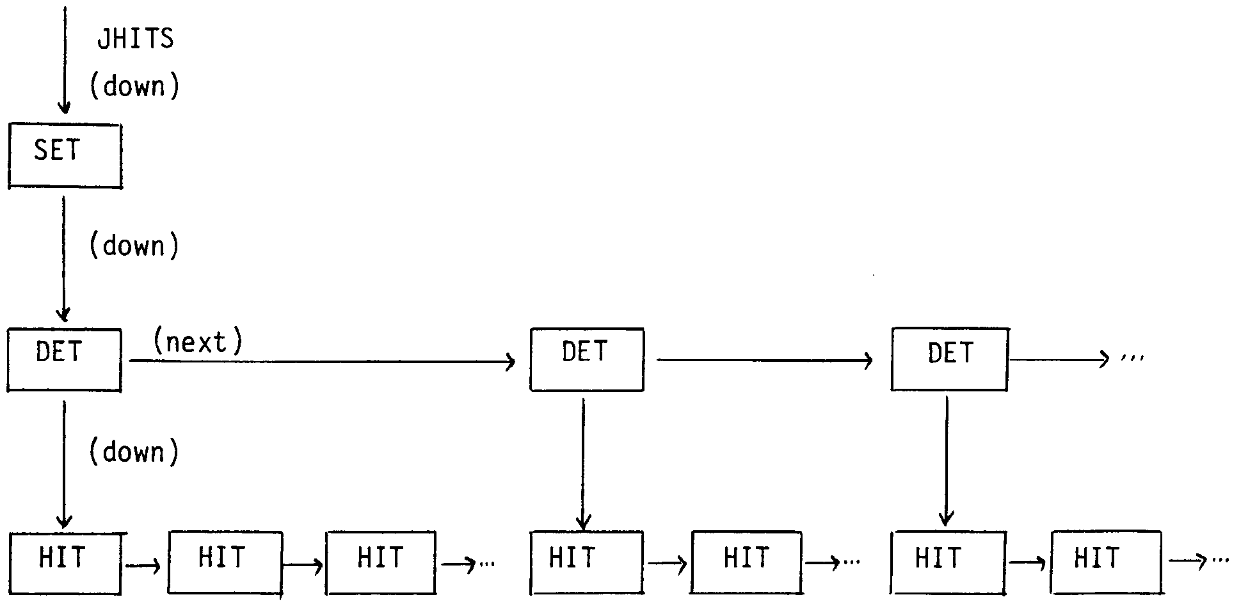

Figure 6: A representation of the hit data structure JHITS in GEANT.

LQ(L − NL − NIO − 1)	IOcb / NL + NIO + 12	I/O control byte / Offset of the bank centre
LQ(L − NL − NIO)	I/O opt. 1	
...	...	Extra I/O descriptor words (0 ≤ NIO ≤ 16)
LQ(L − NL − 1)	I/O opt. NIO	
LQ(L − NL)	Link NL	
...	...	Reference links
LQ(L − NS − 1)	Link NS + 1	
LQ(L − NS)	Link NS	
...	...	Structural (*down*) links
LQ(L − 1)	Link 1	
LQ(L) - - - - - - >	next-link	Address of the *next* bank in a linear structure
LQ(L + 1)	up-link	Address of the supporting bank
LQ(L + 2)	origin-link	Address of the supporting link
IQ(L − 5)	IDN	Numeric bank identifier
IQ(L − 4)	IDH	Hollerith bank identifier (4 characters)
IQ(L − 3)	NL	Total number of links
IQ(L − 2)	NS	Number of structural links
IQ(L − 1)	ND	Number of data words
IQ(L) - - - - - - >	status word	status bits: 1-18 user, 19-32 system
IQ(L + 1)	data word 1	
...	...	Data words
IQ(L + ND)	data word ND	

Figure 7: Format of a ZEBRA bank, from Reference 5.