

# **Digitization of the GEANT4 TWIST Simulation**

Kate Ross  
University of Waterloo  
ID 20098631  
2A Honours Physics

TWIST Research Group, TRIUMF  
Vancouver, British Columbia

Supervisor: Art Olin

May 3<sup>rd</sup>, 2004

# Table of Contents

Table of Contents .....	2
Table of Figures .....	3
1) Introduction.....	4
1.1) TWIST: Physics Purposes and Goals .....	4
1.2) Work Term Goals .....	4
2) TWIST: The Detector.....	5
2.1) Introduction to the TWIST detector .....	5
2.2) Triggering an Event - The Scintillator.....	6
2.3) Drift Chambers and Proportional Chambers: .....	7
2.4) Digital Converters: TDC's and ADC's.....	8
3) Simulating TWIST .....	9
3.1) Why Simulate TWIST? .....	9
3.2) Current Simulation .....	9
3.3) Future of the TWIST simulation: using a new toolkit.....	10
3.4) G3twist vs. G4twist .....	11
Advantages of G4twist.....	11
Disadvantages of G4twist .....	12
3.5) The Task .....	12
4) G3twist Digitization Investigation .....	14
4.1) Summary of G3twist Digitization: .....	14
Storing Hits .....	14
Processing Digitization .....	16
Output .....	17
4.2) Possible Entry Points from GEANT4.....	18
Option 1: Using the G3 hits collection and ZEBRA .....	18
Option 2: Using the G4 hits collection, avoiding ZEBRA .....	18
5) G4twist Hits Collection and Digitization Design .....	20
5.1) Classes .....	20
ChamberHit .....	20
ChamberSD .....	20
ChamberHitsCollecitonG3 .....	20
DigitizationG3 .....	21
5.2) Flow .....	21
Hits Collection.....	21
Digitization Processing and Output.....	24
6) Tests and Results.....	28
6.1) Hits Collection .....	28
6.2) Digitization Processing.....	34
6.3) Output .....	36
7) Conclusions.....	37
8) Recommendations .....	37
APPENDIX A .....	38
APPENDIX B .....	42
References.....	45

## Table of Figures

Figure 1: TWIST detector.....	5
Figure 2: A muon decay event, showing decay positron and neutrinos.....	5
Figure 3: Detector side view .....	6
Figure 4: DC Cell and plane .....	7
Figure 5: Simplified GEANT3 Program Flow.....	15
Figure 6: G3twist Digitization Scheme .....	16
Figure 7: G3twist Output Procedure .....	17
Figure 8: Class Relationships to HC structures, G4twist .....	21
Figure 9: Hits Collection Flow, G4twist .....	22
Figure 10: Digitization Flow, G4twist .....	25
Figure 11: G3twist - DC min drift time.....	28
Figure 12: G4twist - DC min drift time.....	29
Figure 13: G3twist - PC min drift time .....	29
Figure 14: G4twist - PC min drift time .....	30
Figure 15: G3twist - DC leading edge time.....	31
Figure 16: G4twist - DC leading edge time.....	31
Figure 17: G4twist - DC hit positions .....	32
Figure 18: G4twist - PC hit positions.....	33
Figure 19: Leading edge times, before digitization processing – G4twist .....	34
Figure 20: DC leading edge times, after digitization processing – G4twist.....	34

# 1) Introduction

## 1.1) TWIST: Physics Purposes and Goals

TWIST stands for the TRIUMF Weak Interaction Symmetry Test. The TWIST experiment is located in Vancouver, British Columbia at the TRIUMF cyclotron facility. It is an experiment in particle physics, designed to test the validity of the standard model.

TWIST is interested in muon decay, which is an interaction governed by the weak nuclear force. The standard model predicts four parameters, called the Michel parameters, which describe this decay. TWIST's goal is to measure the Michel parameters to an accuracy of at least three times greater than previously achieved. Measuring them to a greater precision will allow definitive conclusions to be made about discrepancies (if any) between what the standard model predicts, and what is actually observed in nature.

It is known that the standard model is not a complete theory. Several theories have been developed as extensions to the standard model. By examining the discrepancies between the standard model and reality with respect to weak interactions, TWIST will allow certain of these theories to be ruled out.

## 1.2) Work Term Goals

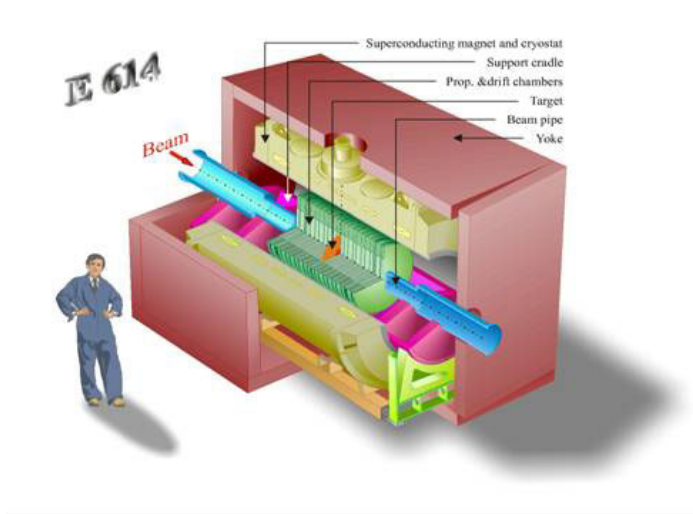
The TWIST experiment relies on its particle detector, as well as a sophisticated simulation of this detector. The current detector simulation, which began development in 1998, was built using a physics simulation toolkit called GEANT3.

GEANT3, developed at CERN in the 1970's, provides the basic tools needed for simulating a particle physics experiment. A more recent version of the toolkit, GEANT4, began development in 1994.

It is proposed that designing a new TWIST simulation using GEANT4 may improve its accuracy. The new version of the simulation, called G4twist, has been in production since 2001, but as of March of 2004 it is still missing a portion of its functionality. The missing portion is the ability to produce output in a form that is equivalent to the real detector data. This process is commonly referred to as *digitization*, and it is an important step in simulating the TWIST detector; without this feature, the simulation data cannot be compared directly to that obtained from the actual detector. Thus, the goal of this work term is to complete the digitization of the new TWIST simulation.

## 2) TWIST: The Detector

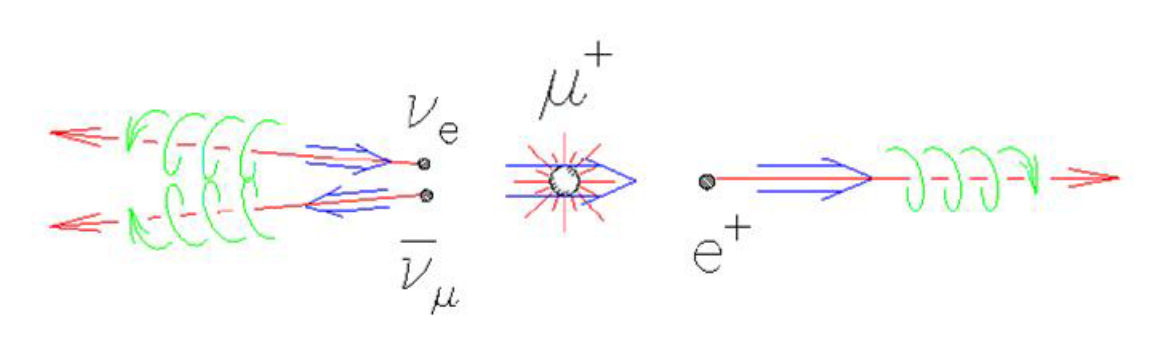
Figure 1: TWIST detector



### 2.1) Introduction to the TWIST detector

A beam of positively charged muons is sent to the TWIST detector (Figure 1) through a beamline provided by TRIUMF's cyclotron. The muon beam enters the detector, which is enclosed in a strong magnetic field generated by a large bore superconducting magnet. The muon follows a helical path through the detector, due to the presence of the field. An individual muon is stopped in the aluminum target, located at the center of the detector. Once the muon has stopped, it decays with a mean lifetime of 2.2 microseconds, producing a neutrino, anti-neutrino and a positron (Figure 2) [1].

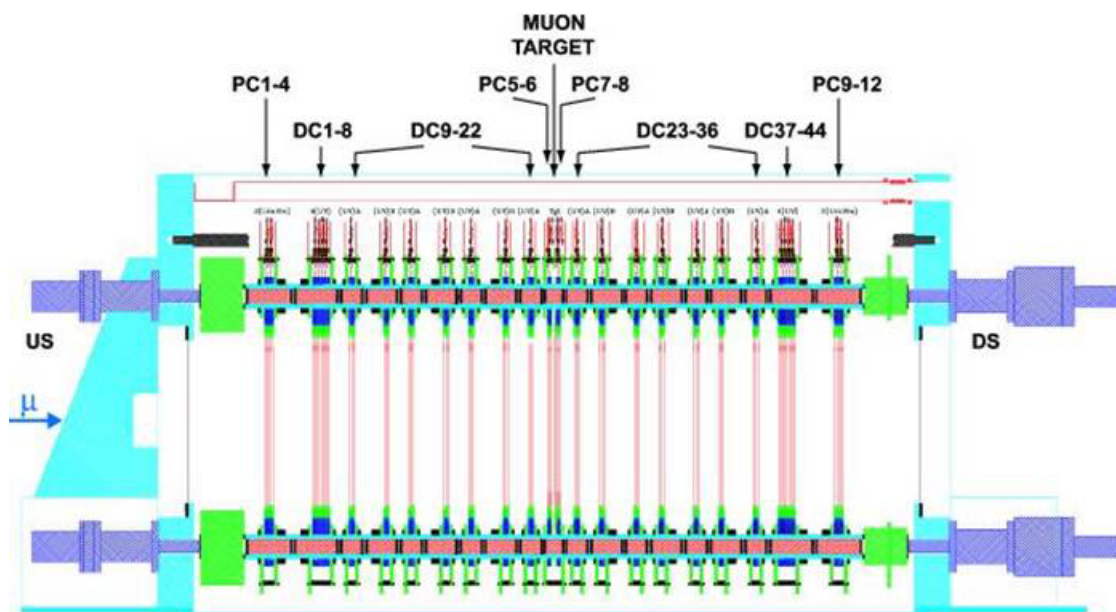
Figure 2: A muon decay event, showing decay positron and neutrinos



The positron that is produced from the decay event also follows a spiral path throughout the detector, and this path is carefully tracked. By analyzing the trajectory of the decay positron, the momentum and angular distributions of the decay event can be determined. From these distributions, the Michel decay parameters can be calculated.

The trajectory of each charged particle is measured by the 56 sense planes that are positioned within the cylindrical magnet. Each sense plane consists of either drift chambers (DC's) or proportional chambers (PC's). The data recorded from each plane can only resolve one direction of the trajectory, depending on the plane's orientation. Thus, the planes are positioned in pairs throughout the detector, resolving two orthogonal directions at each position along the detector. There are 22 pairs of DC planes and 12 pairs of PC planes in the detector, symmetrically divided by the target, which lies at the center of the detector (Figure 3).

Figure 3: Detector side view



## 2.2) Triggering an Event - The Scintillator

The TWIST data depends on the timing of a particle as it passes through the detector. Thus, it is essential to have a reference time that will mark the start of an event. The reference time is defined by a muon traversing a thin scintillator, which is located at the upstream end of the detector stack (closest to the entrance of the

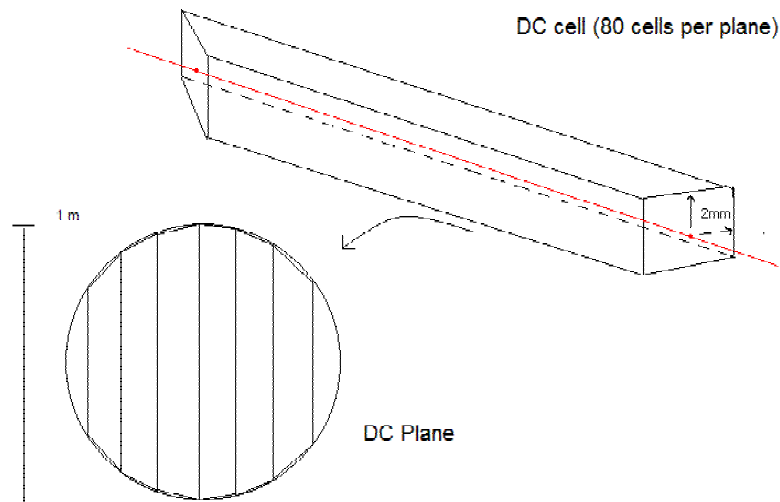
beamline). The electronics are triggered by the first muon to enter the scintillator, and every time recorded thereafter is in reference to the initial trigger time.

### 2.3) Drift Chambers and Proportional Chambers:

The DC and PC's, along with the muon scintillator, are the parts of the detector which are sensitive to the observed particles. They are designed to allow position and time resolution of a particle's trajectory.

The DC and PC planes consist of equally spaced sense wires, each maintained at a high voltage. There are 80 sense wires in each DC plane, separated by 4mm, and 160 sense wires in each PC plane, separated by 2mm. For the sake of analysis, the PC and DC planes are logically divided into *cells*. A cell is a long trapezoidal volume containing one wire (Figure 4).

Figure 4: DC Cell and plane



The chambers are filled with a specific gas, and a high voltage is applied to each sense wire. As a charged particle passes through the chambers, it causes some gas molecules to ionize. Groups of electrons from the ionized molecules form, and are called *clusters*. The charged clusters, due to their small kinetic energy upon ionization, do not leave the chamber with the incident particle. Instead, the clusters drift towards the nearest sense wire, under the influence of the strong electric field. The sense wires are each attached to electronic devices, *time to digital converters* (or TDC's), which allow the time required for the clusters to reach the wires (their *drift times*) to be recorded. Using a map of the electric field in each cell, as well as precise timing information of the incident particle, the drift times of the clusters can be resolved into the position where the ionization occurred - a point in the trajectory of the particle of interest.

The DC and PC chambers, although their designs are similar, are used for two different purposes. To accomplish their specific tasks, two different gases are used, as well as differing electronics. The gas in the proportional chambers (CF<sub>4</sub> – Carbon Tetrafluoride) is called a “fast gas” when compared to the gas in the drift chambers (DME - Dimethyl Ether). Electron clusters take less time to drift to the sense wires in CF<sub>4</sub> than they would in DME. With a smaller drift time (approximately 20ns in PC cells compared to 200ns in DC cells), the particle's *time of flight* can be resolved with more precision. The time of flight, or the time since event trigger, is important because it is useful to determine exactly when the muon and positron traverse the detector. A slower gas is used in the DC chambers, since it results in better position resolution. Thus, the DC's obtain precise positions, and the PC's obtain precise timing information.

Another feature of the proportional chambers is their ability to measure the energy loss of the particle. Because the number of electrons in the cluster is proportional to the amount of energy lost by the ionizing particle, the amplitude of the signal allows measurement of that particle's energy deposit. In order to measure the pulse height, extra electronic devices, *analogue to digital converters* (ADC's), are connected to the PC's that neighbor the target. The energy deposit in those cells can then be measured with the ADC's, and be used as a tool for analysis.

#### **2.4) Digital Converters: TDC's and ADC's**

The TDC's are responsible for converting recorded times to digital signals. The type of TDC used by TWIST accepts one common ‘stop’ time, and up to 8 ‘start’ times for each channel. The stop time is what marks the end of a time interval. The larger the start time, the further away from the stop time it occurred. The stop time is determined by the trigger particle in the scintillator, and the start times are determined by signals on the wire. Another feature of the TDC's is their ability to distinguish *leading edge* and *trailing edge* times. These edges are caused by several groups of electrons which drift to the wire from different distances. The width of a TDC signal is determined by the time difference between the leading and trailing edges. If two or more clusters reach the sense wire at approximately the same time, an *overlap* of the signals occurs and they are combined. The TDC takes the first leading edge and the last trailing edge to be the width of this large signal.



## 3) Simulating TWIST

### 3.1) Why Simulate TWIST?

The nature of the experiment is to measure the differences, if any, between theory and “real physics”. Therefore, it must be clear that the differences observed are in fact due to a divergence of theory from reality, and are in no part due to deficiencies in the experimental setup or method.

The spiral path of the decay positrons produced from this reaction is what holds the key to measuring the Michel decay parameters. The TWIST detector is not 100% efficient at tracking the decay positrons. However, the detector response has been measured very accurately, and efficiency trends are seen for varied angles and energies of the particle of interest. It is hoped to obtain a digital duplicate of the TWIST detector by creating a simulation that incorporates these trends, or “imperfections”. Since each part of the detector is to be exactly replicated, the response of the simulation would ideally be identical to that of the detector. The intentional difference between the detector and the simulation will be how the muon decay occurs. In the simulation, muon decay will be based upon the predictions of the standard model. In the detector, this may or may not be the case.

Thus, the results of the simulation, when compared to real data, will show the true discrepancies between the standard model and reality. In other words, these comparisons will be free of any outside influences caused by the detector response, since these influences should be present in both the simulation and the detector.

### 3.2) Current Simulation

The current version of the TWIST detector simulation has been developed over the course of the past 6 years, with contributions from many collaborators. The simulation was constructed using a well-known physics simulation toolkit, GEANT3 (abbr. G3). Written in FORTRAN, G3 allows users to add their own FORTRAN routines to the basic structure already provided, customizing it for their experiment.

In the simulation, G3 is responsible for sending pre-defined particles through TWIST's geometry. G3 provides the essential physical processes that occur in the TWIST detector, such as ionization, particle scattering, and electromagnetic effects.

There are some factors that are common to every simulation developed using G3. Such factors include the stepping procedure, the particles and physics processes available, and the general organization of the simulation. Each simulation may have a number of primary particles that comprise an *event*. An event is defined by the

passage of these particles through the detector. Several events may be generated to form a *run*. When a run is completed, the simulation terminates.

The TWIST simulation also has many features that are specific to the experiment. The detector is divided into several different volumes, which are carefully positioned to form the detector geometry. Each PC and DC cell is considered a separate volume, and the group of them are arranged to form the PC and DC planes. G3 sends a muon through the detector, updating the particle's current position, time, energy, and momentum, at each step. This procedure, carried out for all primary and secondary particles produced in the simulation, is called *tracking* or *stepping*.

When a particle enters a sensitive part of the detector, every step that it makes within that volume is called a *hit*. Hits can occur in the scintillator, as well as the PC and DC cells. A special procedure is carried out when a hit is made, called *hits collection*, that allows certain attributes of the particle's step to be recorded for later processing. TWIST is interested in information collected for each cell entered (for example, the minimum drift time in that cell), and so a hit record is stored once for every cell, not once for every step. To distinguish between these two types of hits, the record that is kept for a cell will be called a *cell-hit*, which is constructed from several individual *step-hits* (or simply *hits*).

After an event has completed, the hits information is processed in a way that will produce a data file of the same general form and content as that produced by the real TWIST detector. This process is called *digitization*.

TWIST often generates over 10 000 events per run. Each event has specific attributes which are identical throughout the run, such as the number of muons generated per unit area, or the presence and strength of the magnetic field.

The current version of the simulation is still tested and debugged on a continuous basis. The simulation describes TWIST well, and is useful for systematic studies and comparisons with real data.

### **3.3) Future of the TWIST simulation: using a new toolkit**

Efforts to improve the current simulation are still made; a model that better describes the TWIST detector results in smaller systematic uncertainties when simulation output and real data are compared.

One of the efforts made recently to improve the simulation has been to upgrade to the newest simulation toolkit, GEANT4 (abbr. G4). The goal of this endeavor is to determine how well physics processes are modeled by G4 when compared to G3, and if there are any significant advantages to moving to G4 permanently.

The geometry, physics processes, particles, and electromagnetic field are all defined according to TWIST's requirements, in the basis of standard G4 program

design. Some tests have been done to determine the differences between G3 and G4 in terms of available physics processes, such as electromagnetic interactions. However, the simulation is not yet complete, and lacks many features available in G3twist.

### 3.4) G3twist vs. G4twist

#### *Advantages of G4twist*

G4twist has some anticipated advantages over G3twist. The main advantages are its improved physics processes, extendibility, and memory management.

#### *Physics Processes:*

G4 was developed with the goal of improving the available physics processes. Improvements have been made, especially for high-energy processes, which better simulate particle interaction. In particular, the claim is that the electromagnetic processes for muons and electrons in G4 are improvements of those in G3 [3]. This is important in G4twist, since the helical path of the particles in a magnetic field greatly depends on how electromagnetic interactions are simulated. In particular, G4 now employs a more complete model for multiple scattering processes [6]. Multiple scattering occurs in G4twist when a muon interacts with detector materials (such as the Mylar walls of the chambers). It is affected by the cloud of electrons surrounding a nucleus; an interaction which may change its trajectory and cause energy loss.

#### *Extendibility:*

G4 is written in C++, an object oriented language. This is an improvement over G3's base in FORTRAN, which is a procedural language. Because of its object oriented design, G4 provides the ability to extend its capabilities. For example, it is a relatively straightforward task to define new shapes, particles, and physics processes which are not currently provided by G4 [5]. This is not the case with G3. In G3twist, the addition of new physics processes is a lengthy task, requiring the developer to understand many areas of the simulation. For example, the addition of a process that models ion and electron clustering in G3twist required the modification of several existing tracking routines, as well as associated common blocks.

The same process will need to be defined in G4twist, but G4 provides a much simpler method for incorporating it into the framework of the simulation. One must only register the newly defined process with the GEANT4 Process Manager, and no other changes need to be made to the existing code. All the tracking procedures will be aware of the process, and it will use the necessary particles without any intervention from the developer.

#### *Memory Management:*

In the development of G3 at CERN, it was immediately apparent that some form of dynamic memory storage would be needed for this toolkit [4]. The memory storage had to be efficient and flexible, able to store varying amounts of information for each run without the need for editing and recompiling the source code. There is no such type of memory storage available in FORTRAN, so CERN developed their own memory tool called ZEBRA. ZEBRA allows users to create dynamic memory banks, used for the storage and retrieval of data at different points in the simulation.

A more sophisticated language, C++ has built-in dynamic memory allocation abilities. Thus, there is no need for any external memory tools in G4. This is a great advantage for G4twist. The ZEBRA banks are difficult to understand and maintain unless one is very familiar with their structure. They require several operations to be carried out directly within the user's code (initialization, for example). ZEBRA does not provide the level of abstraction that is available with C++ memory allocation. A developer of G4twist does not need to understand the workings of C++'s memory storage, and can easily use it in an effective way. In fact, much of the memory requirements are handled through the use of classes and objects, the basis of C++ programming.

### ***Disadvantages of G4twist***

For all these improvements, G4twist is still lacking in reliability and functionality.

#### *Reliability:*

G4twist is quite new, relative to its predecessor, G3twist. A very significant amount of work has gone into testing and debugging G3twist over the past 6 years. In comparison, very little testing has been done on G4twist, and is not yet known to be reliable.

#### *Functionality:*

G4twist currently lacks a clustering process. However, as noted above, this process can be relatively easily incorporated into the simulation. Most importantly, as of March of 2004, G4twist does not include hits collection and digitization. Without these two areas, the simulation can not be compared to data, making it useless as a production simulation for TWIST.

## **3.5) The Task**

The next step for improving G4twist is to implement hits collection and digitization procedures in the simulation.

There are two general choices, in terms of design, for the digitization of G4twist. The digitization scheme could be entirely rewritten in C++, making use of the various tools available in G4 specific to this task. Another possibility is to use the digitization scheme already in place in the current G3 simulation. This last option involves ‘wrapping’ parts of the FORTRAN code into C++ classes – that is, allowing a method in G4twist to pass required parameters to the G3twist subroutines. The digitization will then be carried out exactly as it is done in the current simulation.

There are obvious advantages to reusing the existing code. Years of work have gone into developing G3twist, and all parts of it have been carefully tested and debugged. Therefore, it is known that the digitization methods will function correctly, as long as they are passed the correct parameters from G4. In this way, a large part of the testing and debugging cycle can be avoided almost entirely.

There are no immediately obvious disadvantages to this method. Wrapping FORTRAN code is, in general, much simpler and less time consuming than translating it. This is especially true when considering that the code has to be translated not only from FORTRAN to C++, but also (and requiring the most thought) from GEANT3 to GEANT4. The two toolkits employ very different organizational principles, and thus the directly translated code would not suffice for a G4 simulation.

Hence, the preliminary task is to investigate the G3twist digitization scheme, decide what code is to be reused, and how that code is to fit into G4twist.

## 4) G3twist Digitization Investigation

### 4.1) Summary of G3twist Digitization:

A simplified flow diagram for a typical GEANT3 simulation can be seen in Figure 5 [2]. The red boxes indicate the areas that contain elements essential to the digitization of the simulation, and are expanded upon in Figure 6.

The subroutine GUSTEP allows the user to assert control over a single step in the current track. This routine is often used for storing hits information in the ZEBRA memory bank, JHITS. When an event has completed, and all hits have been stored, the routine GUDIGI is called, which carries out the processing of the hits information (for example, checks for overlapping signals and combines them), and stores the resulting data in the JDIGI bank. Finally, at the end of a run, GUOUT outputs the contents of JDIGI into a specific format, identical to that of the TWIST detector data files.

The specific digitization scheme for G3twist is mapped in Figure 6.

#### *Storing Hits*

At each step, GUSTEP calls ‘store\_hit\_ch’ and ‘store\_hit\_sc’ to save hits information for the PC and DC cells and the muon scintillator, respectively. These routines extract information about the step, such as position and time of flight, from variables in the tracking common block *gctrack*.

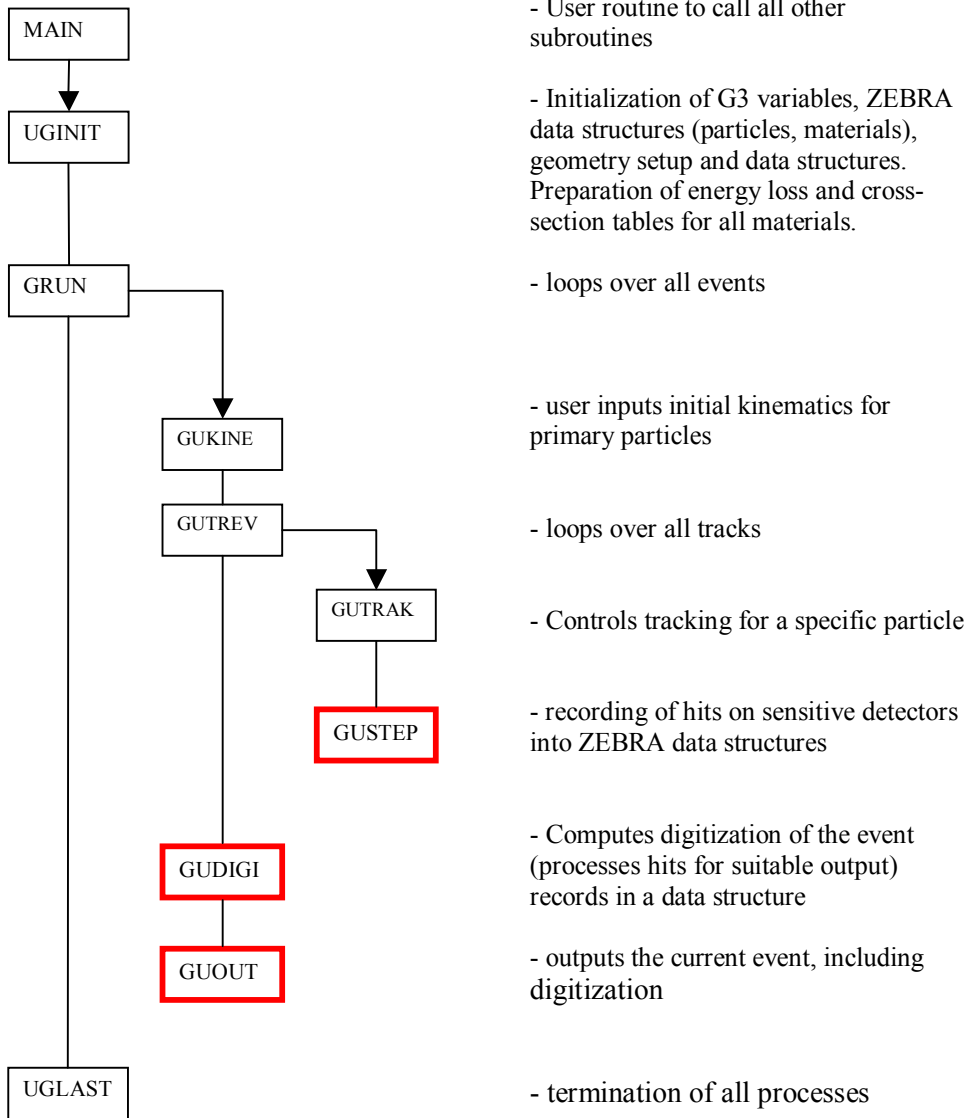
For PC and DC volumes, the drift time of the ion cluster is calculated based on the distance from the sense wire inside the current cell (‘dtg\_gettime’). This is accomplished by accessing a two-dimensional array that contains the calculated drift times for each y-z position within a cell. This array is filled from existing data files called *isochrone maps*. These maps are created by a separate program, which calculates drift times based on material properties of the gas, and the electric field shape in a cell.

After the drift times are retrieved, the trailing edge times for the cluster are calculated (‘sort\_hits\_ch’).

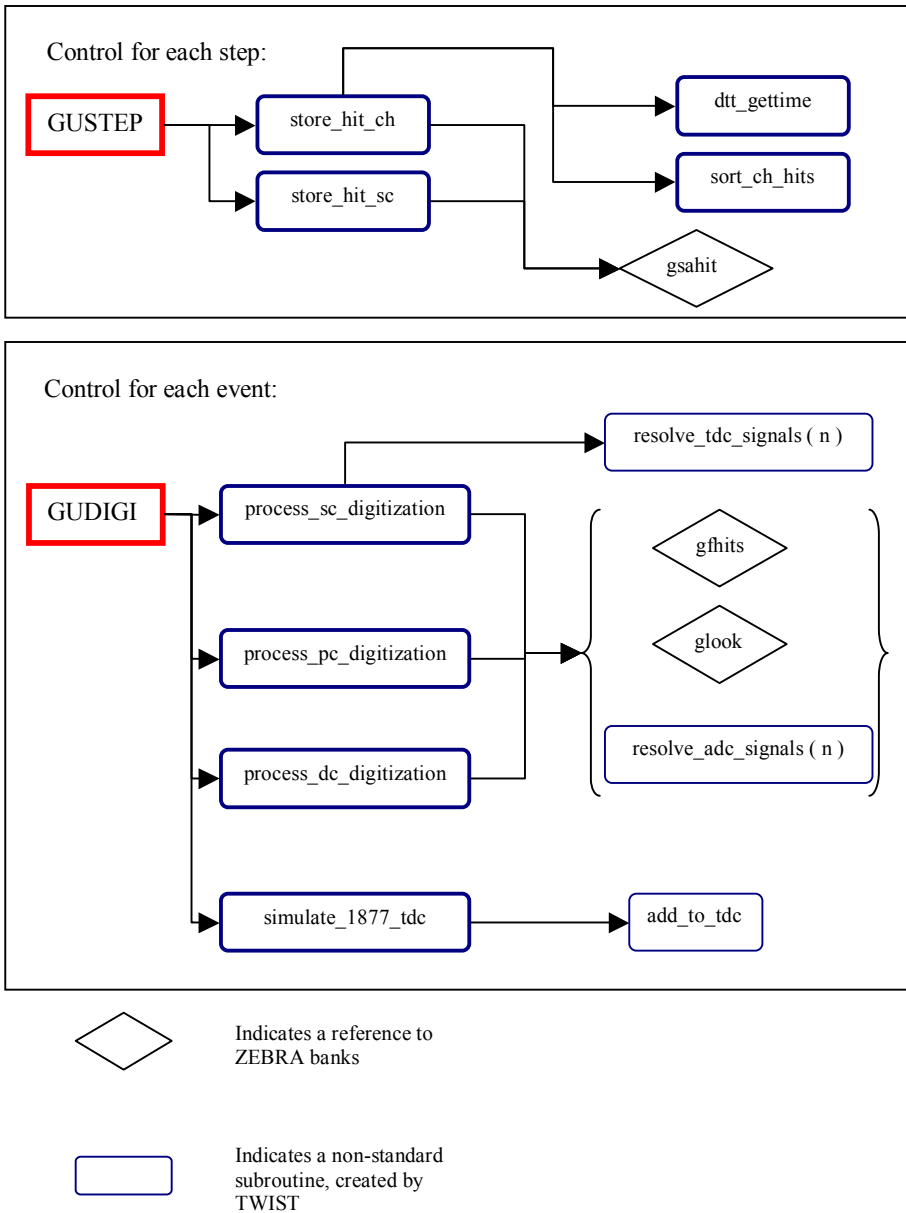
For the scintillator, the first muon to enter the volume resets the time of flight for each other particle present, and thus is the trigger for the event.

The above hits information is stored in an array called 'hits'. Both routines then call GEANT3's 'gsahit', which commits the 'hits' array to the ZEBRA bank JHITS.

**Figure 5: Simplified GEANT3 Program Flow**



**Figure 6: G3twist Digitization Scheme**



***Processing Digitization***

At the end of an event, GUDIGI calls the ‘process\*\_digitization’ routines (where \* is sc, pc, or dc). These routines are responsible for simulating the electronics in greater detail. For example, hits are checked for time overlap (‘resolve\_tdc\_signals’) before being sent to the TDC (‘add\_to\_tdc’).

These three routines must retrieve data from both the JHITS (using ‘gfhits’) and JSET (using ‘glook’) banks in order to process it for digitization.



The JHITS bank is filled by the ‘store\_hit\_ch’ and ‘store\_hit\_sc’ routines described above. The JSET bank is a record of the sensitive geometry structure, including hierarchies of volumes and their identification numbers. This information is defined much earlier in the program flow, at the initialization stage, and greatly depends on the specific detector geometry.

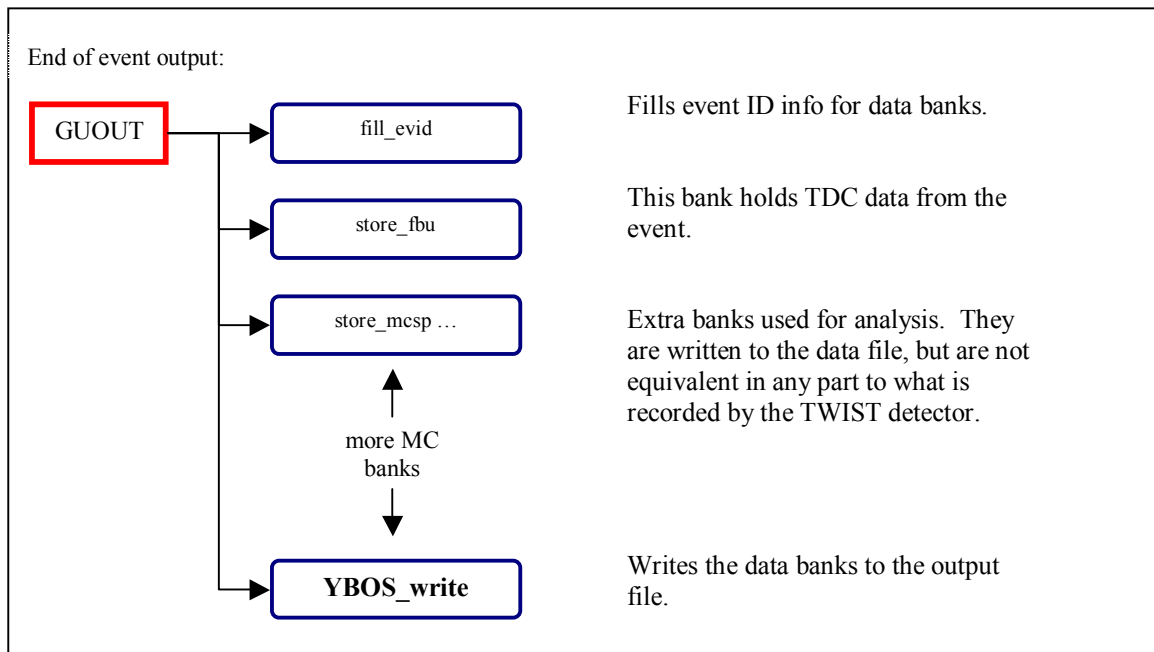
After retrieving hits and volume information, the digitization routines fill arrays that are stored for access by ‘simulate\_1877\_tdc’. This crucial subroutine is responsible for converting leading edge and trailing edge times of the ion cluster, as well as plane and cell numbers, into the bit pattern generated by the LeCroy 1877 TDC’s used by the TWIST detector.

**Output**

Figure 7 shows how the digitization data is outputted to a binary file.

After all of the digitization processing is complete, GUOUT is called. This routine is responsible for storing the processed information in memory banks. G3twist makes use of a type of memory storage that simulates the data tapes used by the detector, called YBOS. There are several YBOS banks that are created by GUOUT, including extra banks found only in simulation data. The main bank is named FBU (filled by ‘store\_fbu’). This bank holds data that is equivalent to what is produced by the TWIST detector. In the final step of digitization, the contents of the FBU bank are written to a data file by a function defined in YBOS (‘YBOS\_write’).

**Figure 7: G3twist Output Procedure**



## 4.2) Possible Entry Points from GEANT4

Two possible entry points were seen from the preceding hits and digitization scheme. In one method, G3 is allowed to handle the hits collection, and the ZEBRA memory banks are used. In the other method, required hits data is retrieved from available G4 hits collection structures, and some G3twist routines are modified slightly. The use of the ZEBRA memory banks is avoided entirely.

### *Option 1: Using the G3 hits collection and ZEBRA*

The entry points in this scheme are the two ‘store\_hit’ routines. At every step within a chamber or scintillator, the routines would be called and provided with the appropriate parameters from G4, such as current step position and time of flight. The ZEBRA banks would have to be initialized early on in the G4 program flow, so that JHITS may be filled correctly by these two routines. Storing and accessing information from these banks would be accomplished by using G3 kernel routines, such as ‘gsahit’ and ‘gfhits’.

While this option does have the advantage of requiring very little translation or modification of existing FORTRAN subroutines, it relies heavily on the ZEBRA memory banks, which are quite complicated, and in this case, problematic.

The problem with the ZEBRA banks lies in the creation of the detector banks JSET and JDET. It may be plausible to recreate the JHITS bank. This would require general initialization to create the bank, and then the ‘store\_hits’ routines to fill it. However, it would be very difficult to recreate the JSET bank. JSET and its related bank, JDET, are initialized and filled by 10 subroutines in G3twist, corresponding to the complete geometry definition. To create these banks in G4, the geometry definition would then have to be carried out twice. The geometry would be defined once for the purposes of G4 and its tracking, and once solely for the creation of the ZEBRA banks. The result would be a mess of geometry definitions, which would be time consuming to modify and would contradict the philosophy of object oriented design.

### *Option 2: Using the G4 hits collection, avoiding ZEBRA*

This option entirely avoids the use of ZEBRA banks. Within G4twist, a new class would be implemented including methods from the G3twist ‘store\_hits’ routines, translated for compatibility with G4 tracking procedures and hits collection structures. The function of the class would be to process and store hits in a manner very similar to that of the JHITS banks. Then, at the end of an event, modified versions of the ‘process\_digitization’ routines would be passed

parameters from these hits collection structures, as well as required geometry information. This would avoid references to JHITS and JSET. These 'process\_digitization' routines would then carry out the remaining digitization exactly as it occurs in G3twist.

Option 2 is the most maintainable, and likely the least time-consuming option available. The challenge then, in terms of design, is to use the standard tools for hits collection in GEANT4 to perform this non-standard procedure.

## 5) G4twist Hits Collection and Digitization Design

The standard design for hits collection and digitization for G4 would not allow the inclusion of G3twist subroutines. The following alternative design allows the use of the flexible hits collection structures of G4, as well as some valuable G3twist routines (Figure 8).

### 5.1) Classes

There are four main classes involved in the hits collection and digitization design. The following is a brief description of each of the four classes: *ChamberHit*, *ChamberSD*, *ChamberHitsCollectionG3*, and *DigitizationG3*. For a template of each class, see Appendix A. For a glossary of their attributes, see Appendix B.

#### *ChamberHit*

This class provides “hit” objects, which store all required information about a hit. For example, the energy deposited, position, and time of hit are associated with an object of this class. Hits data is stored/retrieved by the access functions of this class (e.g. *GetEnergyDeposit()*). The class also has hits collection (HC) structures associated with it, used for storing a group of hit objects.

Access to these HC structures, which are equivalent in purpose to the JHITS bank in G3twist, is the main design challenge in porting hits collection and digitization to G4twist.

#### *ChamberSD*

The methods of this class are implemented when a track deposits energy in a sensitive volume of the detector. PC and DC cells, as well as the muon scintillator, are defined as sensitive volumes. The HC structures are also initialized within a method of this class.

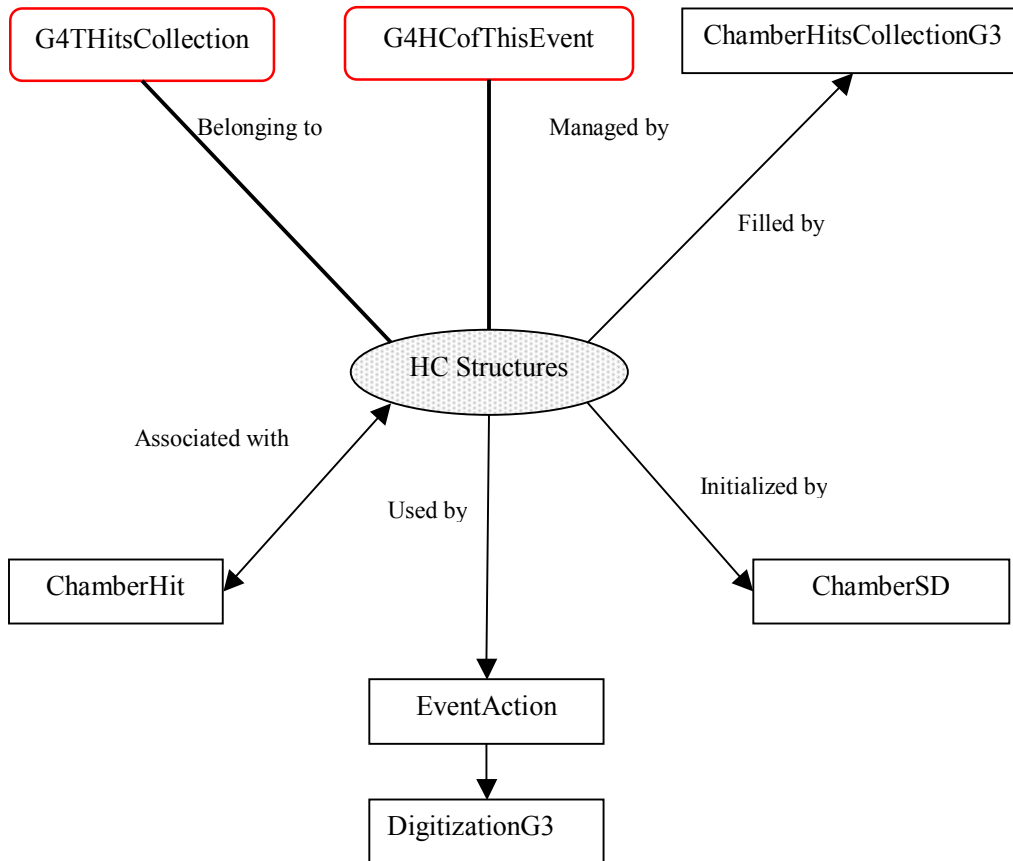
#### *ChamberHitsCollectionG3*

The actual processing of hits occurs here, including calculating drift times and finding the minimum drift distance to the sense wire. This class fills the HC structures initialized by *ChamberSD*; a record is kept for every sensitive volume that has been hit. For example, the record (or cell-hit), includes leading and trailing edge times of each cluster signal.

### ***DigitizationG3***

At the end of an event, this class processes all hits records for digitization. This includes retrieving the records from the HC structures, and converting data to the bit pattern produced by TWIST's TDC. The methods are called from within another user class called *EventAction*.

**Figure 8: Class Relationships to HC structures, G4twist**

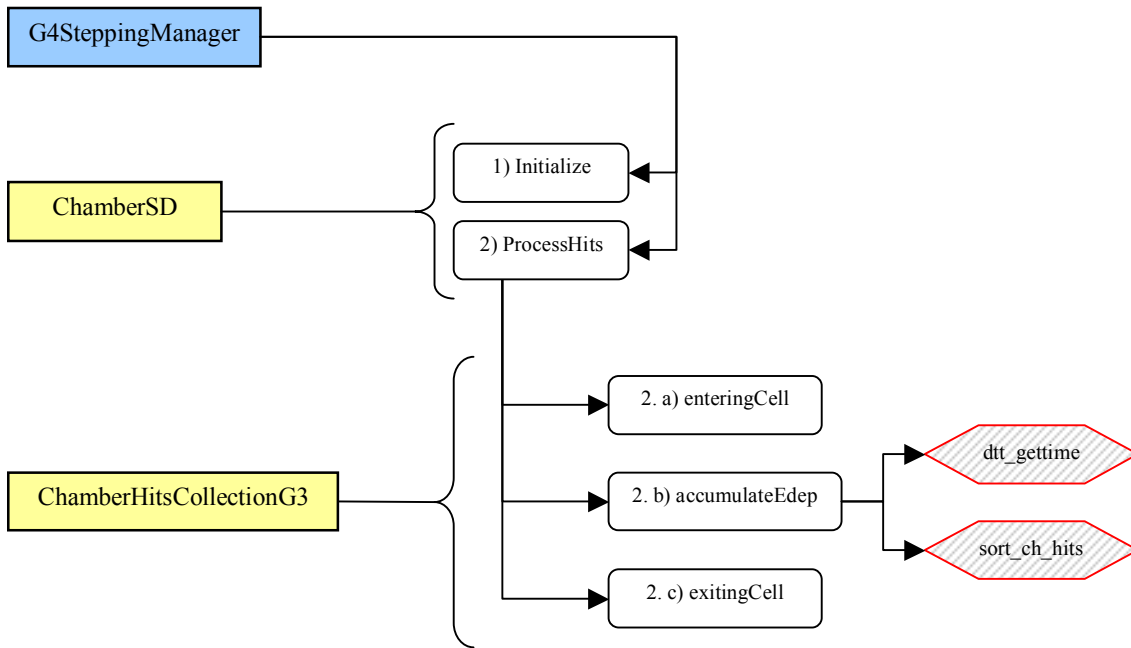



## **5.2) Flow**

### ***Hits Collection***

The following is a breakdown of the hits collection process (Figure 9).

**Figure 9: Hits Collection Flow, G4twist**



 Indicates a FORTRAN routine

### 1) Initialize:

In this method of *ChamberSD*, three HC structures are created and named, one for each type of sensitive volume. They are also given identification numbers, and assigned to the general object HCE (hits collection of this event). HCE manages the HC structures, allowing them to be accessed at the end of the event.

### 2) ProcessHits:

This method is invoked at each step that occurs within the sensitive volumes. It is responsible for determining whether a track is entering, traversing, or exiting a sensitive volume. The corresponding methods, *enteringCell()*, *accumulateEdep()*, and *exitingCell()* of the *ChamberHitsCollectionG3* class, are invoked.

#### a. enteringCell

##### i. If the volume is a DC or PC cell:

This method sets parameters to their initial values for a cell. Such initializations include setting *min\_time* and *max\_time* to large and small numbers respectively, the minimum drift time to zero, and *zwire\_min/zwire\_max* are set to the current point's x-coordinate.

The total energy deposit, as well as total number of hits for the cell, is initialized to zero.

All of these values are likely to be modified by successive invocations of `accumulateEdep()` as the track traverses the cell.

ii. If the volume is the scintillator:

In the TWIST detector, all times recorded by the TDC are relative to the time at which the scintillator is triggered by a muon. In order to simulate event trigger, this method resets the G4 timer to zero once a muon enters the scintillator.

The method also records the time value of every particle entering the scintillator. This value later assists the digitization routines in determining if the event was triggered, since the trigger particle should have an initial time of zero.

**b. accumulateEdep**

i. If the volume is a DC or PC cell:

This method increments the number of hits, recalculates total energy deposit for the cell, and determines the drift time to the sense wire for each step. This latter task is performed by passing the *local* y- and z- coordinates of the current point (in the frame of reference of the specific cell) to the G3twist subroutine 'dtg\_gettime'. This routine then transforms the positions into integer indices, used for retrieving the drift times from the isochrone tables.

The function also determines the minimum drift time based on the drift times of all previous hits in that cell. A deviation is added to the retrieved drift times, based on measured PC and DC resolution, which smears the times in a realistic way.

ii. If the volume is the scintillator:

The total energy deposited in the volume is recalculated.

**c. exitingCell**

This method stores each cell-hit in the HC structures.

i. If volume is a DC or PC cell

The leading and trailing edge times for each step in this cell are calculated based on their previously determined drift times (stored in the `drift_time` array). These leading and trailing edge times are calculated by the G3twist routine 'sort\_ch\_hits'.

The following attributes of the cell-hit are then stored for PC and DC volumes:

- minimum and maximum distances on the sense wire (zwire\_min, zwire\_max),
- space point of the minimum time (min\_point),
- minimum and maximum times on the wire (min\_time/max\_time),
- minimum drift time (min\_dtime),
- leading and trailing edge times (t1[] and t2[]),
- total energy deposit (tot\_edep) ,
- current cell and plane numbers (fChamberNb, motherCopyNo),
- chamber name (aPVname),
- particle ID (iPart),
- number of hits (nhits)

ii. If volume is the scintillator

The following attributes of the cell-hit are stored for the SCIN volume:

- current volume number (fChamberNb)
- current mother volume number (motherCopyNo)
- particle ID (iPart)
- total energy deposit (tot\_edep)
- initial time of flight (tofin)

### ***Digitization Processing and Output***

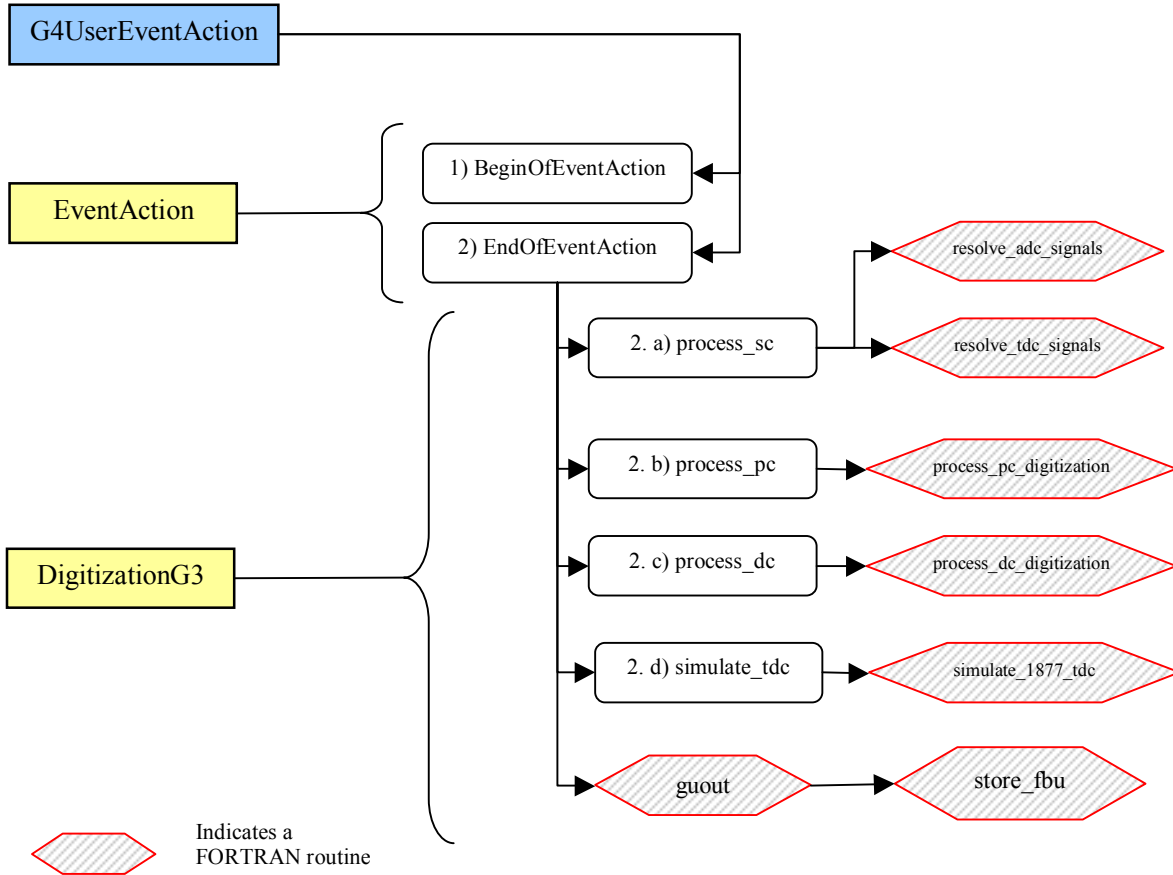
The digitization design in G4twist was constructed as a gateway to the G3twist ‘process\_digitization’ routines. These routines store ADC and TDC information in the arrays ‘hit\_times\_sc’, ‘hit\_times\_pc’, and ‘hit\_times\_dc’. These arrays contain information that will be converted to the bit pattern generated by TWIST’s TDC, in the routine ‘simulate\_1877\_tdc’.

Due to difficulties with references to the JSET bank in “process\_sc\_digitization”, this routine has been completely translated to C++. The two routines “process\_pc\_digitization” and “process\_dc\_digitization” were simply modified to be compatible with GEANT4 without being translated.

The following is a breakdown of the digitization processing stage (Figure 10).



Figure 10: Digitization Flow, G4twist



**1) BeginOfEventAction**

At the start of an event, this method retrieves the HC structures.  
The three HC structures are then available for use in EndOfEventAction.

**2) EndOfEventAction**

Before any methods are invoked, this is where the user may print out the contents of the HC structures for debugging. It is also where preliminary histograms may be generated for testing purposes, before any digitization is performed.

Each of the following methods, belonging to the class *DigitizationG3*, is then invoked:

- a. **process\_sc**

This method loops through all scintillator volumes (although there is now only one, there is the possibility for future extension) and creates a two-dimensional array, 'signals', based on hit information.

'signals' contains, for every hit:

- time of TDC hit, taken from the HC structure's record of 'tofin'
- width of TDC hit, calculated based on above value
- time of ADC hit, calculated from the hit's 'tofin'
- width of ADC hit, calculated base on above value

The 'signals' array is stored in a memory location that is shared between the C++ classes and the FORTRAN subroutines. It is then used by the G3twist routine 'resolve\_adc\_signals', which determines the start energy and total energy for the ADC hit.

If there is more than one hit stored in this volume, 'resolve\_tdc\_signals' is called in order to combine overlapping time signals, if they exist, as well as check for dead regions of the wire.

A two-dimensional array 'hit\_times\_sc', stored in the shared memory location, is then filled. It contains:

- scintillator ID number (this value is always 1)
- leading edge time
- trailing edge time
- time and width of the first ADC hit

#### **b. process\_pc**

A two-dimensional array is filled with hit information from the PC hits collection structure.

For each cell in a given plane, the following information is passed to 'process\_pc\_digitization' through the two-dimensional 'hits' array:

- minimum position (position of minimum drift time) x, y, and z coordinates.
- particle ID
- min/max time
- min/max position along the wire
- total energy deposit
- number of hits
- leading and trailing edge times

A second array ('cells') contains cell numbers in the order they are stored in 'hits'. This array is also passed to 'process\_pc\_digitization', and is used internally to sort hits based on cell number, removing the dependency on the JSET bank.

The FORTRAN routine ‘process\_pc\_digitization’ has been modified such that it no longer retrieves information from JHITS and JSET, but instead is passed the required information through the ‘hits’ and ‘cells’ arrays. It then carries out its task of dealing with the multiplexing of various PC wires, and also resolves the ADC and TDC signals (dealing with signal overlap, for example). The resulting data is placed in the common array ‘hit\_times\_pc’, to be used in the routine ‘simulate\_1877\_tdc’.

**c. process\_dc**

The same procedure as above is carried out, with the hits information retrieved from the DC hits collection structure instead.

The FORTRAN routine ‘process\_dc\_digitization’ is called for each plane, accepting the ‘hits’ and ‘cells’ array. This routine is responsible for resolving the TDC signals, and filling ‘hit\_times\_dc’ for later use in the routine ‘simulate\_1877\_tdc’. Note that there are no multiplexed wires in the DC planes, as well as no ADC connections, making this routine simpler than ‘process\_pc\_digitization’.

**d. simulate\_tdc**

This method simply calls the G3twist routine ‘simulate\_1877\_tdc’, passing it the current event number.

This FORTRAN routine is responsible for converting all processed hit data (stored in ‘hit\_times\_sc’, ‘hit\_times\_pc’, ‘hit\_times\_dc’) to the bit pattern generated by TWIST’s TDC.

**e. guout**

The output process is entirely wrapped from G3twist. The YBOS banks are initialized and the data file is opened for writing at the start of a run, in the class *RunAction*. GUOUT, a G3twist routine, is called directly from within EndOfEventAction in the *EventAction* class. It calls the necessary routines responsible for filling and writing out YBOS banks with information from the ‘hit\_times’ arrays. Currently, only the detector bank FBU is created and written to a binary data file through the routine ‘store\_fbu’ (see Figure 7: G3twist Output Procedure).

## 6) Tests and Results

In order to test the new hits collection and digitization code, a series of comparisons between G3twist and G4twist were made. For the following plots, each describes a run with 1000 events, consisting of positrons with an initial kinetic energy of 20 MeV. The magnetic field is switched off. There is a linear distribution in x-y for the positrons' vertices (within a 5cm radius), and the z-coordinate for each vertex is -81.0 cm.

The maximum step length is set to 450 microns in G4twist. In G3twist, the step length is determined by the ion clustering process at each step, but has a mean value of approximately 300 microns.

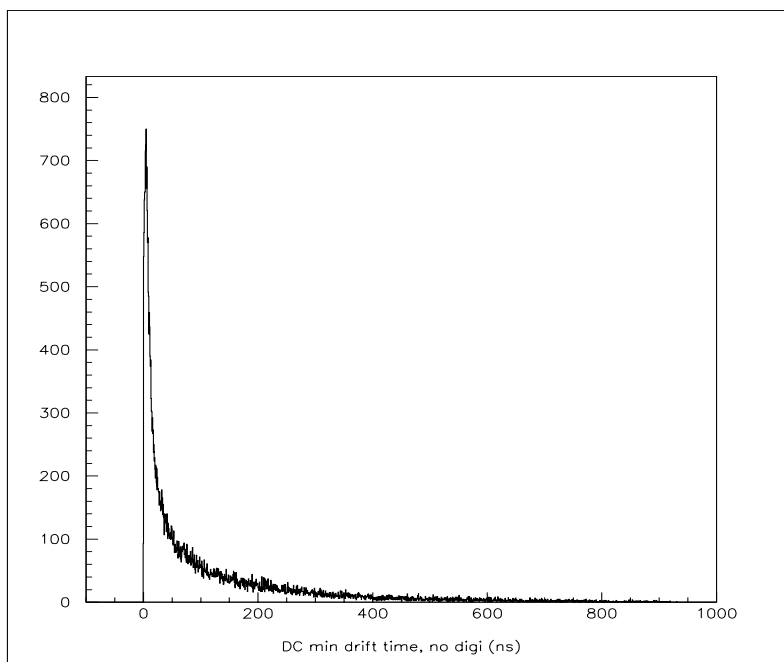
The data are plotted both before and after the digitization routines are called, in order to test the two areas of code separately.

### 6.1) Hits Collection

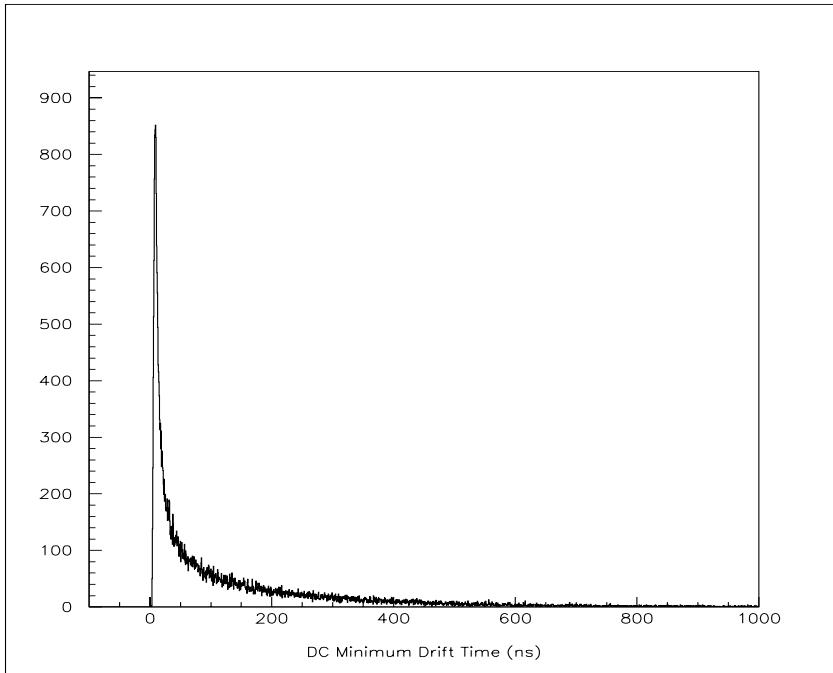
#### 1) *Minimum Drift times*

The following figures (Figure 11 - Figure 14) are histograms of the minimum drift times for each DC and PC cell.

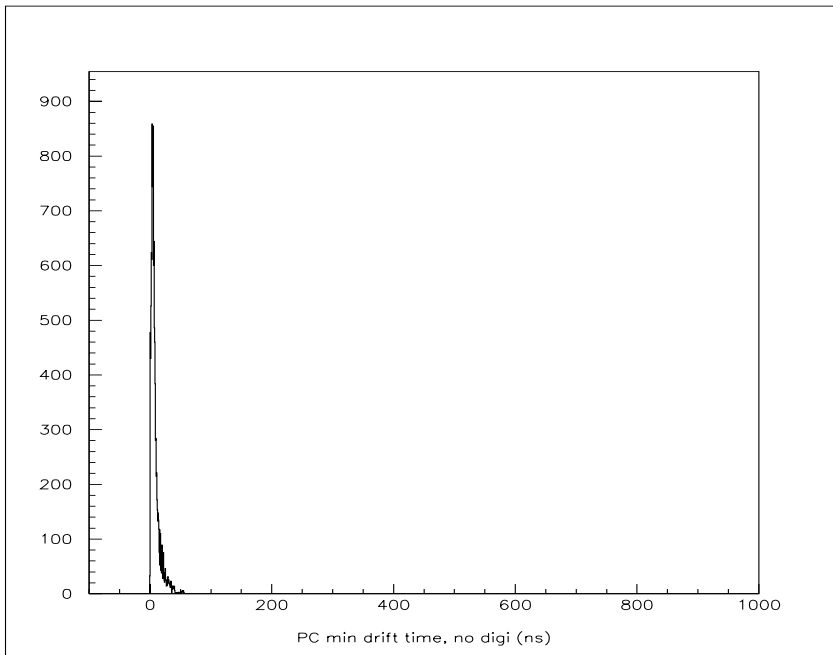
**Figure 11: G3twist - DC min drift time**



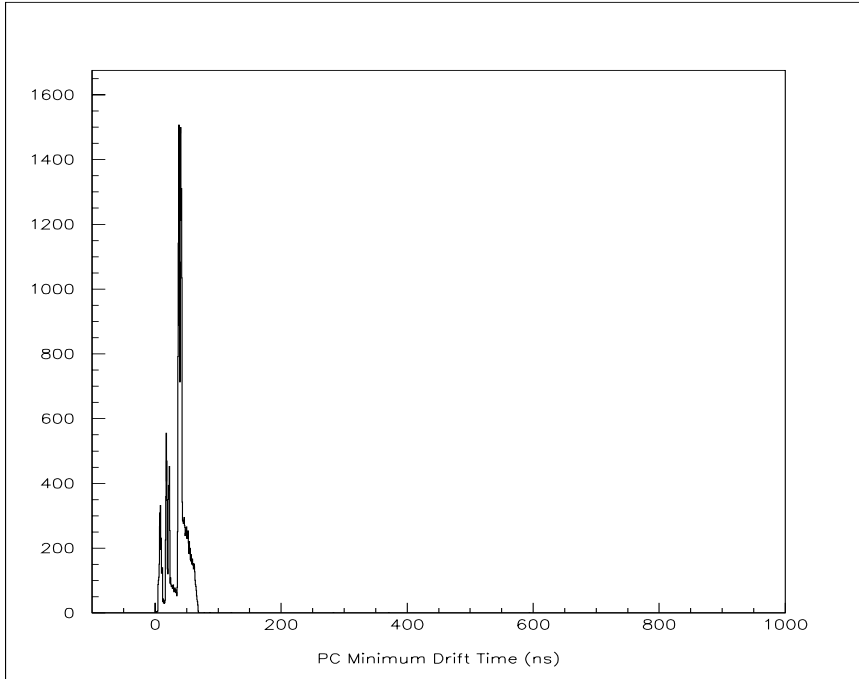
**Figure 12: G4twist - DC min drift time**



**Figure 13: G3twist - PC min drift time**



**Figure 14: G4twist - PC min drift time**



***Discussion:***

The distribution of minimum drift times for DC and PC cells is expected to be a smooth curve with one sharp peak around zero ns. It is also expected that the range of minimum drift times for PC cells will be smaller than that of the DC cells. This is due to the smaller area of the PC cells, since the drift time depends on the distance from the wire. It is also due to fast gas used in the PC cells, which causes all drift times to be shorter.

These trends are clearly displayed by the G3twist plots. The G4twist plot of DC minimum drift time is adequately similar to that produced by G3twist.

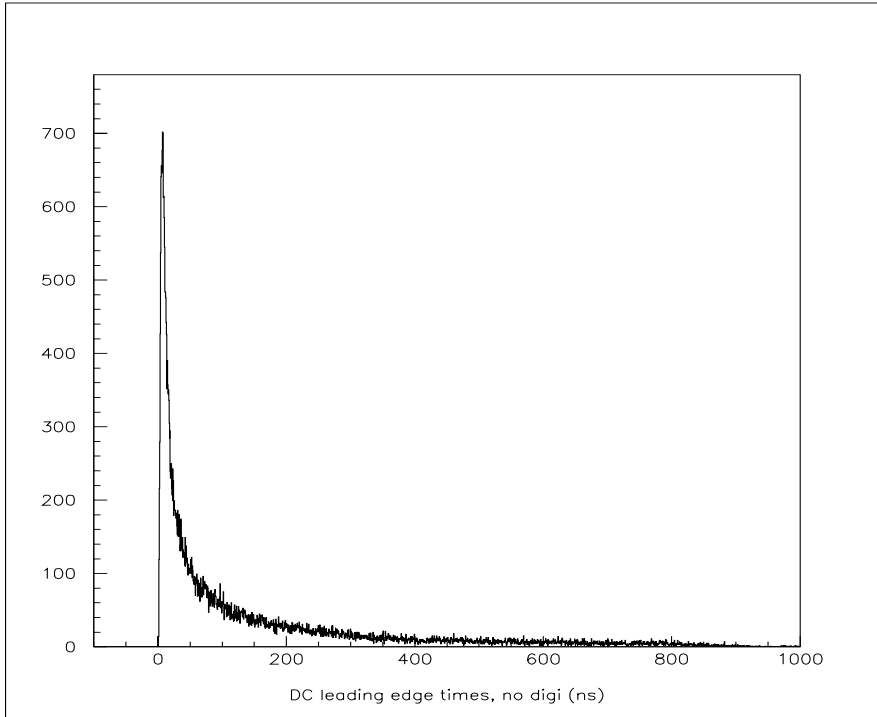
However, the distribution of PC minimum drift time produced by G4twist shows significant differences compared to that produced by G3twist. It is not a smooth curve, but instead shows several peaks.

In order to investigate this discrepancy in the PC cells, the distribution of leading edge times for *every* step-hit (not just the minimum time) was studied for both types of cells.

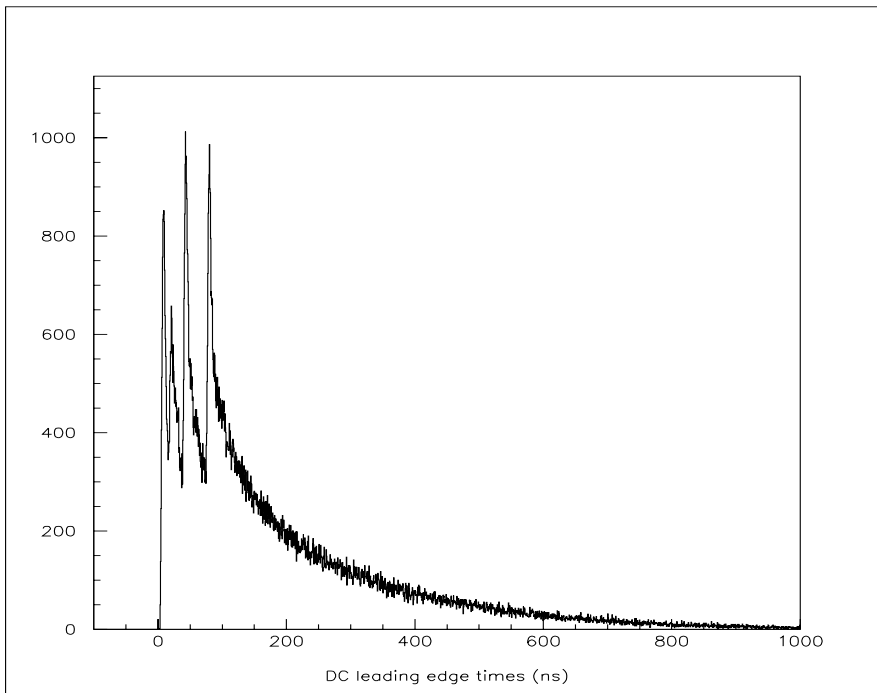
## 2) Leading edge times

The following figures (Figure 15 and Figure 16) are histograms of the leading edge time for each step-hit in all DC and PC cells.

**Figure 15: G3twist - DC leading edge time**



**Figure 16: G4twist - DC leading edge time**



**Discussion:**

The distribution of leading edge times is expected to be a smooth curve similar to that of the minimum drift times. As displayed by the G3twist distribution for DC cells, only one peak is expected. The G4twist distribution shows multiple peaks in leading edge time, indicating a problem within the code.

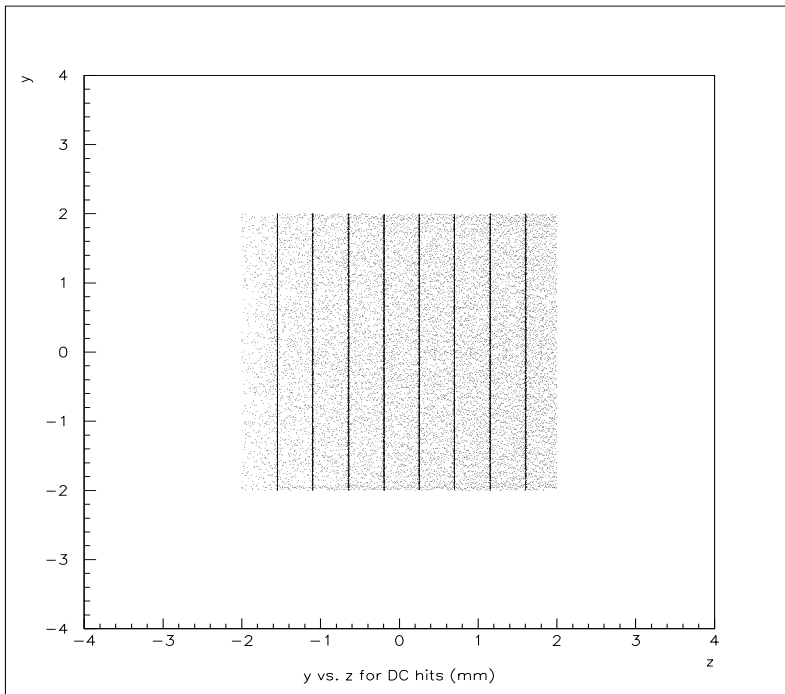
One possible source for this error could arise from the steps that the particle takes through the cell. If the particle takes the same specific steps each time it enters a DC cell, the leading edge times for that step will repeat for every cell, and cause the sharp peaks observed in Figure 16. These peaks would not necessarily occur in the minimum drift time distribution, as the minimum time may not be produced by one of these repeated steps.

If the error lies in the stepping procedures, a two-dimensional plot of the position of all hits within the cells would confirm it. It would not show a random distribution of points, but an obvious pattern instead. Such two-dimensional plots were constructed for both types of cells in G4twist.

**3) Y-Z hit coordinates**

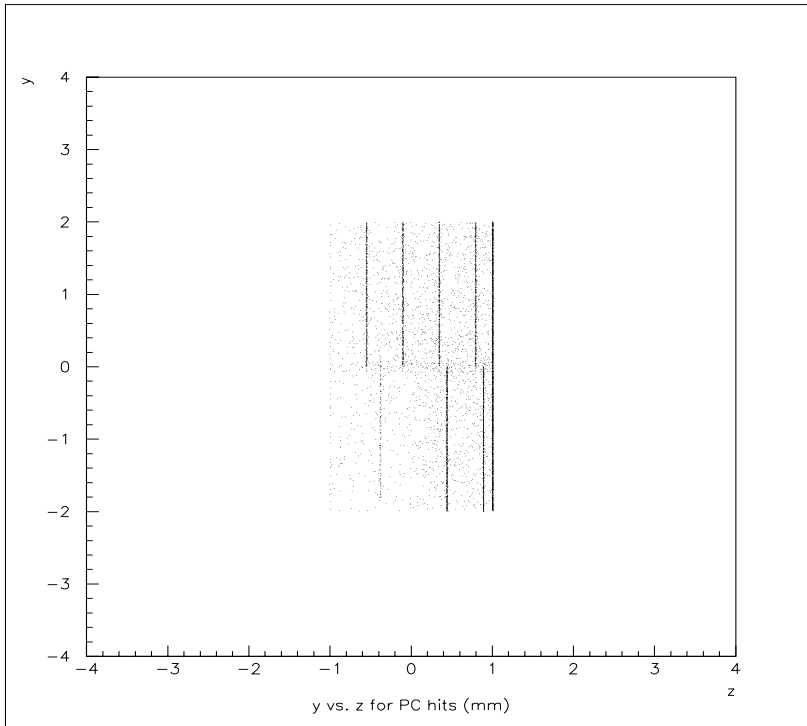
The following figures (Figure 17 and Figure 18) show the y vs. z coordinates of all step-hits in all DC and PC cells in G4twist. In the frame of reference of the cells, the wire lies along the x-axis, and thus the following figures represent superimposed end-views of all cells.

**Figure 17: G4twist - DC hit positions**





**Figure 18: G4twist - PC hit positions**



***Discussion:***

Obvious lines are present in these distributions, instead of the expected random distributions. This confirms that the problem is indeed a stepping error.

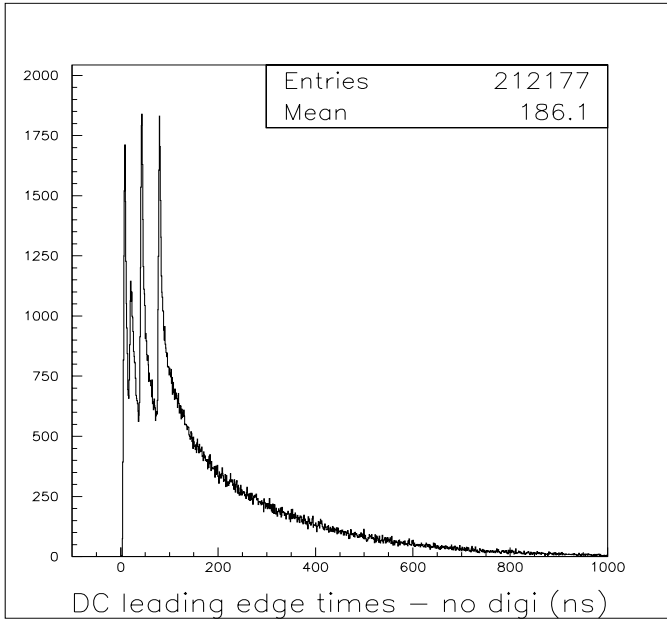
Upon closer inspection, it can be seen that the lines in the DC cells are separated by approximately 0.4 mm. The separation corresponds to the maximum step length of 450 microns that was set for this run, indicating that the problem may lie in G4's implementation of user step limits. The toolkit should smear the step positions slightly, based on the step limit, to avoid similar problems, but it does not appear to do this.

The distribution for PC cells is more puzzling. The dark lines are seen to be asymmetrical across the z axis. This may suggest a bug in the geometry, as it only occurs within the PC cells. If there is a bug in the PC geometry, there may be a similar one in the DC cells, which in fact causes the stepping errors.

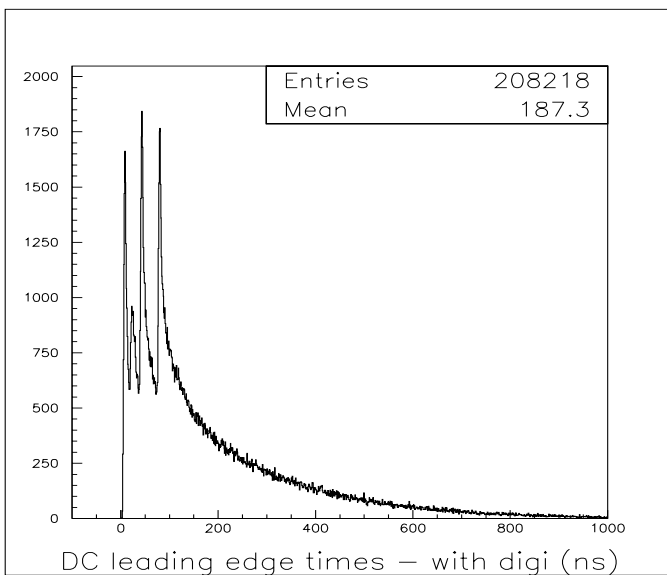
## 6.2) Digitization Processing

The following figures show the leading edge times in DC cells for G4twist. Figure 19 is a histogram of leading edge times before digitization processing (similar to Figure 16), and Figure 20 shows the distribution after hits have been processed for digitization. Note: digitization processing refers to the implementation of the methods “process\_sc”, “process\_pc”, “process\_dc”, and “simulate\_tdc”. Output to a data file using “guout” is not a factor in these distributions.

**Figure 19: DC Leading edge times, before digitization processing – G4twist**

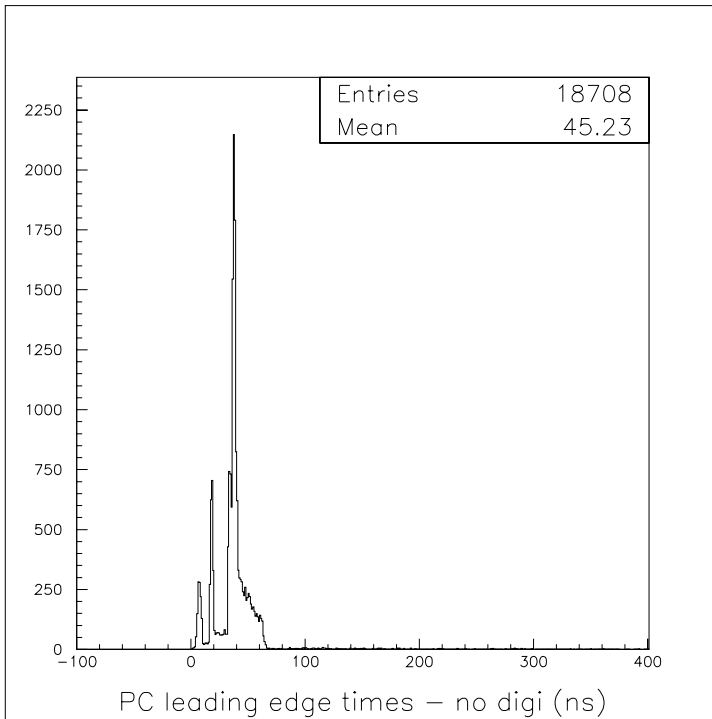


**Figure 20: DC leading edge times, after digitization processing – G4twist**

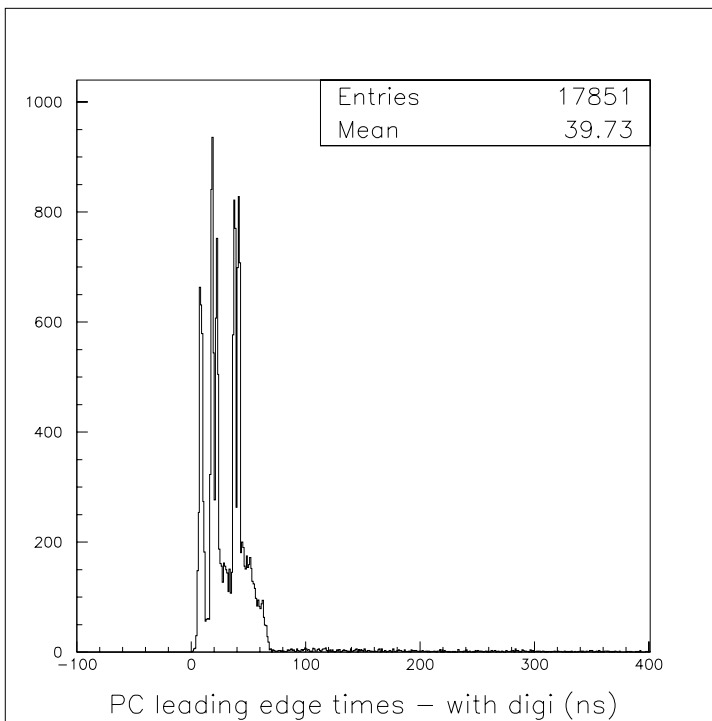


The following figures (Figure 21 and Figure 22) show the leading edge times in PC cells in G4twist, before and after digitization processing.

**Figure 21: PC leading edge times, before digitization processing – G4twist**



**Figure 22: PC leading edge times, after digitization processing – G4twist**



### ***Discussion:***

The similarity of the above figures is as expected. After digitization processing, the same shape in the time spectra should be seen, with slight variations. For the plots of DC leading edge time (Figure 19 and Figure 20) there are small differences in the shape of the histograms, especially noted at the 25ns peak, which is smaller following digitization processing. In the plots of PC leading edge time (The following figures (Figure 21 and Figure 22) show the leading edge times in PC cells in G4twist, before and after digitization processing.

Figure 21 and Figure 22) it can be seen that the large peak at 50ns has been dramatically reduced.

The main difference in these distributions is the number of entries in each. For DC cells, there are 212 177 hits included in the histogram before digitization. After digitization, this number is reduced to 208 218 hits. For PC cells, 18 708 hits are reduced to 17 851. The number of hits is expected to decrease after digitization, due to the TDC overlap procedures which are carried out in 'resolve\_tdc'. This routine combines several hits if their leading and trailing edges overlap.

Although these distributions are still incorrect when compared to G3twist, as noted in the above section, they do serve as preliminary evidence that the digitization routines have functioned correctly.

### **6.3) Output**

An attempt was made to run standard analysis procedures on the G4twist output data file. This test showed that the file is of the correct format, but it does not contain any TDC data. This error may be due to incorrect storage of the 'hit\_times' arrays, which hold the data that is to be written to the YBOS FBU bank. It may also be an error in wrapping the YBOS\_write routine, which performs the actual file output.

## 7) Conclusions

A method for digitizing G4twist has been employed that allows the reuse of some G3twist subroutines, but that also avoids any reliance on ZEBRA dynamic memory structures. All dynamic memory requirements are handled effectively by C++ structures, while still successfully incorporating wrapped/translated G3twist routines.

The hits collection and digitization processing methods of G4twist have been tested against those in G3twist. It has been seen that a significant error is introduced in G4twist by the stepping procedures. This error is influenced by the maximum step length set by the user, a process implemented within GEANT4. When this error is taken into account, it can be seen that the observed errors in G4twist time spectra are not due to the hits collection and digitization processing methods.

A binary data file of the correct format is successfully created in the output stage for each run. However, this file is not yet useful for analysis, and data from G4twist cannot yet be compared to data from the TWIST detector. The problem can be attributed to an output error, and not to a bug in digitization processing.

## 8) Recommendations

It is recommended that before any further comparisons are completed, an analysis of the stepping error be performed. A two-dimensional histogram of y vs. z for all steps inside the detector (similar to Figure 17) at *global* scope, should be made, using a small number of events. Such a test may show patterns emerging not only in the cells, but in all volumes, which may indicate a deeply-rooted problem in GEANT4. If it does not show a pattern, a more detailed look at the PC and DC cells, especially in terms of geometry definition, may be in order.

It is also recommended that an investigation of how ‘guout’ retrieves data from the ‘hit\_times’ arrays be undertaken. An error in this area would cause the FBU bank to be filled incorrectly, which would account for ‘empty’ data files.

Lastly, an effort should be made to include an ion clustering process in G4twist, based on what has already been implemented in G3twist.

When these areas are improved, useful analysis can then be run on G4twist data, and detailed comparisons can be made to both G3twist and real data.

## APPENDIX A

### G4twist Class Templates

#### ChamberHitsCollectionG3

##### Methods:

```
Public:
    enteringCell() : void
    accumulateEdep() : void
    exitingCell() : void
```

##### Attributes:

```
Private:
    DChitCollection : pointer G4THitsCollection
    PChitCollection : pointer G4THitsCollection
    SChitCollection : pointer G4THitsCollection

    fStepPoint : constant pointer G4StepPoint
    fTrack : pointer G4double
    fEdep : G4double
    fPartName : G4String
    fChamberNb : G4int
    fIDtype : G4int

    nhits
    tot_edep : static G4double

    dPos : static G4ThreeVector
    min_point : static G4ThreeVector
    zwire_min : static G4double
    zwire_max : static G4double

    min_time, max_time, min_dtime : static G4double
    drift_time[LOCAL_HITS_MAX] : static G4double
    t1[LOCAL_HITS_MAX] : static G4float
    t2[LOCAL_HITS_MAX] : static G4float

    tofin : static G4double
```

## ChamberHit

### Methods:

Public:

```
    TWISTChamberHit * GetHit() {return this;}
```

```
//Access methods for all private  
//attributes are also available.
```

### Attributes:

Global:

```
    LOCAL_HITS_MAX = 200; //max hits in a cell
```

Public:

```
    t1 [LOCAL_HITS_MAX] : G4float array  
    t2 [LOCAL_HITS_MAX] : G4float array
```

Private:

```
    //for all volumes  
    //-----  
    trackID : G4int  
    fPartID : G4int  
    fPVname : G4String  
    chamberNb : G4int  
    planeNb : G4int  
    edep : G4double  
    pos : G4ThreeVector  
  
    //for DC and PC cells only  
    //-----  
    nhits : G4int  
    fMin_time : G4double  
    fMax_time : G4double  
    fZwire_min : G4double  
    fZwire_max : G4double  
  
    //for SCINT only  
    //-----  
    tofin : G4double
```

## ChamberSD

### Methods:

Public:

```
Initialize(G4HCofThisEvent * ) : void  
processHits(G4Step *, G4TouchableHistory *) : G4bool  
EndOfEvent (G4HCofThisEvent *) : void  
  
clear() : void  
drawAll() : void  
printAll() : void
```

### Attributes:

Private:

```
DChitCollection : pointer G4THitsCollection  
PChitCollection : pointer G4THitsCollection  
SChitCollection : pointer G4THitsCollection  
  
G3HC : pointer ChamberHitsCollectionG3
```

## EventAction

### Methods:

Public:

```
BeginOfEventAction(G4Event * ) : void  
EndOfEventAction(G4Event * ) : void
```

### Attributes:

Private:

```
scCollID : G4int  
pcCollID : G4int  
dcCollID : G4int  
  
G3digi : DigitizationG3
```



## DigitizationG3

### Methods:

#### Public:

```
process_sc() : G4bool  
process_pc() : G4bool  
process_dc() : G4bool  
  
simulate_tdc(G4int eventNo) : void
```

### Attributes:

#### Global:

```
MAX_SIGNALS = 24;  
nhmax_sc = 25;  
nhmax_pc = 1000;  
nhmax_dc = 2500;
```

#### Private:

```
DChitCollection : pointer G4THitsCollection  
PChitCollection : pointer G4THitsCollection  
SChitCollection : pointer G4THitsCollection  
  
nhits_SC : G4int  
nhits_PC : G4int  
nhits_DC : G4int  
  
hits_pc[nhmax_pc][100] : G4float array  
hits_dc[nhmax_dc][100] : G4float array  
  
cells_pc[nhmax_pc] : G4int array  
cells_dc[nhmax_dc] : G4int array
```

## APPENDIX B

### GLOSSARY of ATTRIBUTES

#### *At Global Scope:*

<b>LOCAL_HITS_MAX</b>	The maximum number of hits in a cell. It is a constant used as the maximum size for arrays in <i>ChamberHitsCollectionG3</i> . (200)
<b>MAX_SIGNALS</b>	Maximum number of signals in one cell (25)
<b>nhmax_sc</b>	The maximum number of hits for the scintillator in one run (25)
<b>nhmax_pc</b>	The maximum number of hits for the PC's in one run (1000)
<b>nhmax_dc</b>	The maximum number of hits for the DC's in one run (2500)

#### *ChamberHitsCollectionG3:*

<b>DChitCollection</b>	Stores hits for all DC cells until the end of an event.
<b>PChitCollection</b>	Stores hits for all PC cells until the end of an event.
<b>SChitCollection</b>	Stores hits for all SC cells until the end of an event.
<b>fStepPoint</b>	Pointer to G4StepPoint, used to access information about the current step (position, current volume, etc). Can refer to either the 'pre step point' or the 'post step point', depending on how the object of this class is constructed.
<b>fTrack</b>	Pointer to G4Track, used to access information about the current particle. Also used to reset the time to zero when event is triggered (fTrack->SetLocalTime()), and retrieve the current time of flight (fTrack->GetLocalTime()).
<b>fEdep</b>	Energy deposit for the current step.
<b>fPartName</b>	Name of current particle.
<b>fChamberNb</b>	Copy number of the current cell (equal to 1 for the scintillator).
<b>fIDtype</b>	Type of current volume (1 = DCEL, 2 = PCEL, 3 = SCIN).
<b>nhits</b>	Total number of hits in one cell.
<b>tot_edep</b>	Total energy deposit in one cell.
<b>dPos</b>	Position in daughter coordinate system (reference frame of the current cell). The sense wires for PCEL and DCEL lies along the x-axis at (x, 0, 0).
<b>min_point</b>	Position at which the minimum total time occurs (total time is drift time plus time of flight).
<b>zwire_min</b>	minimum position along the sense wire (x-axis) for a cell.
<b>zwire_max</b>	maximum position along the sense wire (x-axis) for a cell.
<b>min_time</b>	Minimum total time (drift time plus time of flight).
<b>max_time</b>	Maximum total time (drift time plus time of flight).
<b>drift_time[]</b>	Stores total times (drift time plus time of flight) for all hits in a

	cell.
<b>t1[]</b>	Stores all leading edge times for ion clusters within a cell.
<b>t2[]</b>	Stores all trailing edge times for ion clusters within a cell.
<b>tofin</b>	Initial time of flight for any particle entering the scintillator (will be zero for the trigger particle).

ChamberHit:

<b>t1[]</b>	Stores all leading edge times for ion clusters within a cell.
<b>t2[]</b>	Stores all trailing edge times for ion clusters within a cell.
<b>trackID</b>	track number which particle belongs to.
<b>fPartID</b>	particle number for the hit (corresponding to GEANT3 particle codes) 1) gamma 2) e+ 3) e- 4) neutrino 5) mu+ 6) mu-
<b>fPVname</b>	Name of volume in which hit occurred (PCEL, DCEL or SCIN).
<b>chamberNb</b>	Number of cell in which cell-hit occurred.
<b>planeNb</b>	Number of plane in which cell-hit occurred.
<b>edep</b>	total energy deposit for the cell-hit.
<b>pos</b>	min_point for the cell-hit (see min_point in <i>ChamberHitsCollectionG3</i> ).
<b>nhits</b>	total number of hits in the cell-hit.
<b>fMin_time</b>	see min_time in <i>ChamberHitsCollectionG3</i> .
<b>fMax_time</b>	see max_time in <i>ChamberHitsCollectionG3</i> .
<b>fZwire_min</b>	see zwire_min in <i>ChamberHitsCollectionG3</i> .
<b>fZwire_max</b>	see zwire_max in <i>ChamberHitsCollectionG3</i> .
<b>tofin</b>	see tofin in <i>ChamberHitsCollectionG3</i> .

ChamberSD:

<b>DChitCollection</b>	See DChitCollection in <i>ChamberHitsCollectionG3</i> .
<b>PChitCollection</b>	See PChitCollection in <i>ChamberHitsCollectionG3</i> .
<b>SChitCollection</b>	See SChitCollection in <i>ChamberHitsCollectionG3</i> .
<b>G3HC</b>	Pointer to an object of <i>ChamberHitsCollectionG3</i> .

EventAction:

<b>scCollID</b>	ID number (in G4's <i>SDManager</i> ) for scintillator's hits collection structure.
<b>pcCollID</b>	ID number (in G4's <i>SDManager</i> ) for PC's hits collection structure.
<b>dcCollID</b>	ID number (in G4's <i>SDManager</i> ) for DC's hits collection structure.
<b>G3digi</b>	Pointer to an object of the class <i>DigitizationG3</i> .

*DigitizationG3:*

<b>DChitCollection</b>	see DChitCollection in <i>ChamberHitsCollectionG3</i> .
<b>PChitCollection</b>	see PChitCollection in <i>ChamberHitsCollectionG3</i> .
<b>SChitCollection</b>	see SChitCollection in <i>ChamberHitsCollectionG3</i> .
<b>nhits_SC</b>	total number of cell-hits in SChitCollection.
<b>nhits_PC</b>	total number of cell-hits in PChitCollection.
<b>nhits_DC</b>	total number of cell-hits in DChitCollection.
<b>hits_pc[][]</b>	2D array containing hits information for a particular PC plane. Passed to the G3twist subroutine "process_pc_digitization".
<b>hits_dc[][]</b>	2D array containing hits information for a particular DC plane. Passed to the G3twist subroutine "process_dc_digitization".
<b>cells_pc[]</b>	1D array containing the PC cell numbers for the cell-hits in one plane. (In the same order that the cell-hits appear in the array hits_pc [[]])
<b>cells_dc[]</b>	1D array containing the DC cell numbers for the cell-hits in one plane. (In the same order that the cell-hits appear in the array hits_dc [[]])

## References

- [1] Brewer, Jess H. Positive Muon Decay. [Online: <http://musr.physics.ubc.ca/~jess/musr/cap/mudk.htm>] (2004)
- [2] Bruyant, F. GEANT3 User's Guide: Simplified Flow Diagram. [Online: <http://wwwasdoc.web.cern.ch/wwwasdoc/geantold/H2GEANTBASE01.html>] CERN, Geneva 1993.
- [3] CERN Application Software Group. GEANT4 Physics Reference Manual. [Online: <http://wwwasd.web.cern.ch/wwwasd/geant4/G4UsersDocuments/UsersGuides/PhysicsReferenceManual/html/node88.html>] (2003)
- [4] CERN Application Software Group. The ZEBRA System. [Online: [http://wwwasdoc.web.cern.ch/wwwasdoc/zebra\\_html3/node6.html#SECTION03100000000000000000](http://wwwasdoc.web.cern.ch/wwwasdoc/zebra_html3/node6.html#SECTION03100000000000000000)] CERN, Geneva 1995.
- [5] dell'Acqua, Andrea. GEANT4 Web Lectures: Introduction to GEANT4. [Online: <http://www.wlap.org/atlas/umich/geant4/2001/Lectures.html>] (2001)
- [6] Quraan, Maher. GEANT3 to GEANT4 Comparisons: Energy Loss and Multiple Scattering. TRIUMF, May 23, 2003.