GEANT steps into the future

S. Giani, S. Ravndal, M. Maire
CERN, Geneva, Switzerland

Abstract

The algorithms developed and the software used to produce a computer generated movie on HEP simulations are explained. The GEANT program is used to design and visualize the detectors and to simulate LHC events. The PAW program is used to store such events and to produce a sequence of frames for animation. The PAW ntuples then allow the temporal sorting of the particles' positions and the storing of the GEANT images and kinematics.
Several techniques to optimize the search for the closest boundary in very large geometrical data bases are discussed. The geometrical modeling and the tracking algorithm developed in GEANT 3.21 are presented and explained in detail.
Finally, the DRDC P59 proposal to re-design GEANT for an object-oriented environment will be presented. This R&D project will test fundamentally the suitability of the object-oriented approach for simulation in HEP, where performance is a crucial issue.

1 Introduction to the GEANT package

The GEANT program [1] simulates the passage of elementary particles through matter. Originally designed for high-energy physics (HEP) experiments, it has today found applications also outside this domain in areas such as medical and biological sciences, radio-protection and astronautics.

The principal applications of GEANT in high-energy physics are:

- the transport of particles through an experimental setup for the simulation of detector response;
- the graphical representation of the setup and of the particle trajectories.

The two functions are combined in the interactive version of GEANT. This is very useful, since the direct observation of what happens to a particle inside the detector makes the debugging of the program easier and may reveal possible weakness of the setup.

In view of these applications, the system allows the user to:

- Describe an experimental setup by a structure of geometrical volumes. A medium number is assigned to each volume by the user. Different volumes may have the same medium number. A medium is defined by the so-called tracking medium parameters, which include reference to the material filling the volume.
- Accept events simulated by Monte Carlo generators.
- Transport particles through the various regions of the setup, taking into account geometrical volume boundaries and physical effects according to the nature of the particles themselves, their interactions with matter and the magnetic field.
- Record particle trajectories and the response of the sensitive detectors.
- Visualise the detectors and the particle trajectories.

The program contains *dummy* and *default* user subroutines called whenever application-dependent actions are expected.

It is the responsibility of the user to:

- code the relevant user subroutines providing the data describing the experimental environment:

- assemble the appropriate program segments and utilities into an executable program;
- assemble the appropriate data records which control the execution of the program.

The GEANT program is currently used all over the world by many hundreds of users to simulate HEP detectors. It consists of about 200000 lines of code (in more than 1300 routines) and it is estimated that its development represents at least 50 man-years of work, spread over more than 15 years. GEANT is widely used because of its geometry/tracking and visualization packages which enable the simulation of many different experimental setups. The fact that GEANT permits users to plug routines for handling experiment-dependent code (such as the geometrical description of the detector, and the details of the digitisation) into the infrastructure of experiment-independent code (such as tracking, description of materials and particle data structure) is another major factor in its widespread usage. The overall support of the package by a dedicated team, headed initially by R.Brun, then by F.Carminati and now by S.Giani, working in close collaboration with external collaborators and with the CERN program library team, also contributes to make GEANT a de-facto standard in HEP. The GEANT team is mainly responsible for the basic kernel functions of geometry, tracking and electromagnetic physics, but it also integrates user contributions, notably several dealing with the simulation of physical processes. A Motif-based Graphical User Interface based on KUIP [2] is used by both GEANT and PAW [3].

Requests for ever increasing functionality in the program have been handled traditionally by developing code and writing interfaces to external libraries. Moreover, many users developed code to simulate additional physical processes and then requested that such functionality be included in the GEANT library. This led to a continual increase in the size of the program and resulted in a rather difficult maintenance job.

2 Video presentation

Here we explain how the video "GEANT steps into the future" was produced at CERN. The GEANT tracking system allows the simulation of the particles' interaction with matter and the visualization of detectors. It is described more in detail in the next section. The PAW n-tuples are used in this case for a time-driven visualization of physical events.

2.1 The GEANT tracking applied to detector visualization

The tracking of particles through a geometrical data structure is the key functionality of GEANT. At every step of the particle, the program must find the volume where the particle is located and the next boundary it will cross. This can take about 60% of the total simulation time (even for detectors described in an optimized way); therefore, a very sophisticated logic has been introduced to minimize the time spent for the search in the geometrical tree. As a result, we have a very powerful tool to navigate in extremely complex geometrical structures (corresponding to huge geometrical data bases). Such a tool has been found to be essential in the visualization of the experimental set-ups. Based on this new GEANT tracking, a set of routines doing light processing has been written explicitly to visualize the detectors (it is useful also to visualize the results of Boolean operations). Visible light particles are tracked throughout the detector until they hit the surface of a volume declared not transparent; then, the intersection point is transformed to the screen coordinates and the corresponding pixel is drawn with a computed hue and luminosity. Automatically, the colour is assigned according to the tracking medium of each volume and the volumes with a density less than that of air are considered transparent; alternatively, the user can set color and visibility for the desired volumes. Parallel view and perspective view are possible and the detector can be cut by three planes orthogonal to the x,y,z axis or/and by a cylindrical surface. Different

light processing can be performed for different materials (from matt plastic to metal to glass). Parallel light or a point-like-source can be selected and an extra light source can be positioned in space with the desired intensity. Realistic renderings have been produced for detectors made of a few thousands objects as well as for detectors made of several millions of objects. Each frame produced for the film (half a million pixels) takes on average less than 2 minutes on a HP735. A parallel version of the program, giving an almost linear speed-up with the number of processors, is also available.

2.2 The PAW n-tuples applied to visualization of events

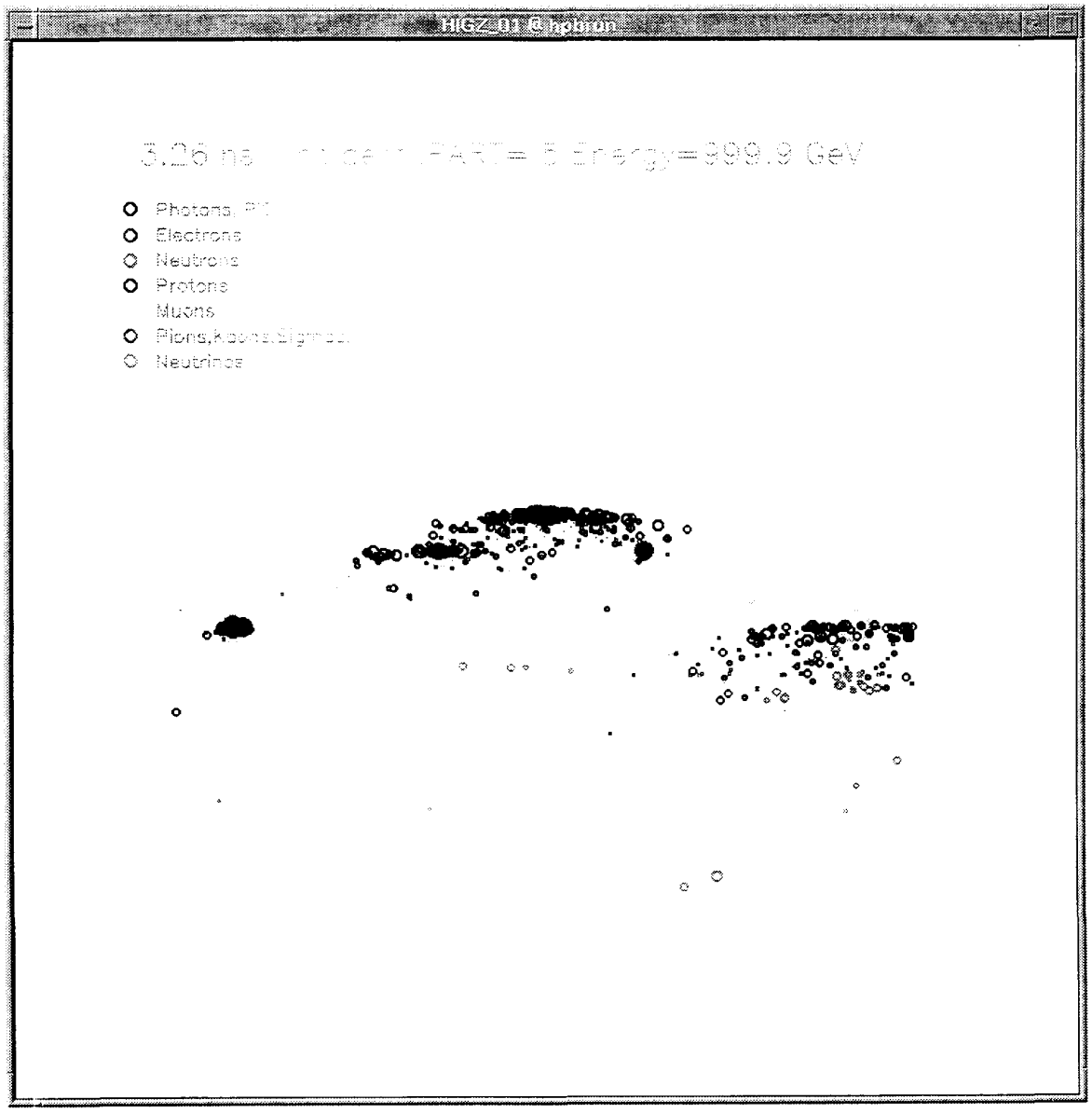

Figure 1

A new way of visualizing tracks and events has also been developed. Instead of drawing the particles' trajectories one after another, all the particles (represented by points or circles, the size depending on the energy) are plotted at the same time in the space position they occupy at a given instant (Fig. 1). So the visualization is driven by time steps of 20

pico-seconds. This technique enhances the understanding of the physical processes and also helps to visualize the behaviour in time of hadronic and electromagnetic particles. This was made possible by saving in a row-wise ntuples (RWN) the position and time of flight of all the particles at every GEANT step during the simulation. After that, the positions must be sorted in time. This happens in 4 phases: an array is filled with the numbers of particles existing at each time sequence of 20 pico-seconds; memory-resident RWN are created with a buffer size proportional to the numbers of particles for each time sequence; they are filled via the subroutine call HFN, which implicitly sorts them in time; finally, a column-wise ntuple (CWN) is created and one row is filled for each time sequence (each row contains a variable number of particles).

2.3 Interactive visualization and analysis

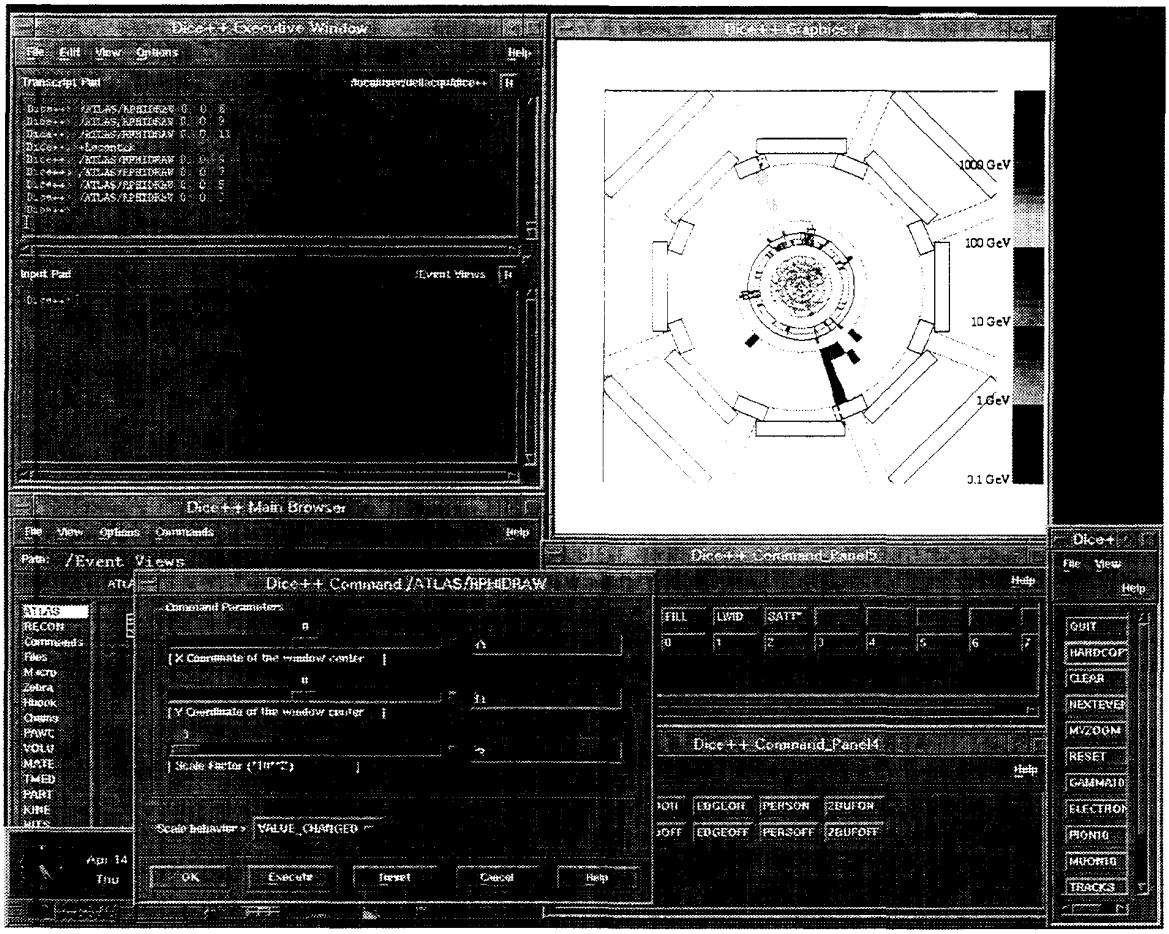

Figure 2

A large portion of the film has been dedicated to the graphical user-interfaces (GUI) of the programs Geant++ and Paw++ (GEANT and PAW using a OSF/Motif-based GUI) (Fig. 2). An extensive set of interactive facilities has been demonstrated. In particular, the GEANT data structures are considered as KUIP browsable classes and their contents as 'objects' on which one can perform actions (the GEANT commands). According to the Motif conventions, an object can be selected by clicking the left button of the mouse (when the cursor is on the icon representing that object). Clicking then on the right button of the mouse, a menu of possible commands will appear (double clicking on the left button the first action of this list will be executed); the selected command will perform

the relative actions on the selected object. Such actions (like drawing, for example) can be executed either directly or via the use of an automatically opened Motif panel. Objects drawn in the graphics window can be 'picked' as well (for example, volumes, tracks, hits); clicking the right button when the cursor is on the selected object, a menu of possible actions on that object is displayed. It is also possible to define Motif panels containing buttons corresponding to the most frequently used commands.

2.4 The video production

The Integrated Video system (by Integrated Research) is used for the sequence generation and display. These sequences can be stored on disk in various formats. For this film we have used "movie" files, which allow long sequences (45s), at the expense of the quality of the pictures. This format has been used for the showers development sequences. For high quality pictures (short sequences of 10s maximum), "rgb" files are used instead. This format has been used for ray-traced detector sequences (a total of about 800 frames have been produced). Besides the sequence generation, the program "ivideo" has been used for the mixing of pictures (detectors in the background and the shower on top), and for the generation of smooth transitions between sequences. All the titles have been produced with the system "Ntitle" (from XAOS Tools). The hardware used to produce the film is an SGI Indigo-2 machine with a 24-bit graphics display, 160Mb of memory and 2Gb of disk space. A "Galileo Box" has been used to generate a video signal (PAL, NTSC) from the full screen or from a portion of the screen (PAL or NTSC format). The "Galileo box" was driven with the SGI software "videoout".

3 Tracking techniques for simulation and visualization packages

The GEANT tracking system allows the simulation of particle interactions with matter and the visualization of detectors. In this section the tracking algorithm developed in the current version of GEANT to optimize the search for the closest volume is described in detail. In a particle's tracking system, at every step, the program must find the volume where the particle is located and the next boundary it will cross. In this process, there are two eventual time-expensive kinds of computation to be taken into account: the computation of the distance to a very large number of volumes (possible candidates) and the computation of the distance to a single but very complicated volume.

In the first case, the geometrical data structure might contain a huge number of volumes and, although the distance-computation to each of them can be quite cheap, the time-consuming computation comes from the fact that a lot of volumes have to be checked.

In the second case, only a few volumes might have to be checked, but the computation of the distance to each of them can be very time very expensive due to their complexity; the best example is the case of objects bound by NURBS (Not Uniform Rational B-Splines), for which the distance-computation requires several iterations (such as the Newton method) or refinements of the surface (Oslo method).

In HEP detector simulation we are normally faced to a problem of the first type, so we will concentrate on the methods to reduce the number of volumes to be checked at each particle's step, assuming that our volumes are bounded by surfaces limited to the second order.

3.1 Standard tracking techniques

Having to find the closest boundary to a particle, the simplest technique consists of computing its distance to all the possible volumes and take the shortest one.

One might argue that we can find more efficient methods. Actually, first of all one can bound more complicated objects by simple boxes or spheres and compute the distance to them before computing the distance, for example, to a complex polygonal shape: if the 'bounding box' is not intersected, the polygon does not need to be checked.

Similarly, one can extend the technique to create 'bounding volumes' which encapsulate a given number of volumes: each of them, in turn, can be a bounding volume for another set of objects, and so on. This allows us to create a hierarchical data structure which can be very effective in limiting the number of volumes to be checked.

Further, one can introduce binary searches in the branches of such a hierarchical tree, making the time proportional to the logarithm of the number of volumes, rather than being linear with them. Incidentally, we have noticed that, for up to about 10 objects, the hardware may perform a linear search faster than a binary search.

Alternatively, it is possible to split the space in cells limited by the boundaries of the embedding volumes (non-uniform grids), again recursively. The embedding volumes can actually be defined just with the purpose of optimizing the non-uniform grid. Once again, as the cells are all different, the time has a logarithmic dependence from the number of volumes per node.

In cases when even a logarithmic dependence is not satisfactory, one can try to build uniform grids, splitting the space in identical cells and associating to each cell the list of volumes intersecting with them. Then at every step the particle will have to deal only with the volumes associated with the current cell and, given the uniformity of the grid, it is straightforward to find the next cell along the particle trajectory. It is clear that the more granular is the grid, the more times the particle will have to step to cell boundaries, but fewer volumes will have to be checked at every step. Moreover, one can fall in the so-called 'paradox of the ball in the empty stadium', stepping in many empty cells far from the volumes which have determined the granularity of the grid.

3.2 GEANT3.21 tracking: the virtual divisions

The tracking of particles through the geometrical data structure is the key functionality of GEANT. A new logic has been introduced to minimize the time spent for the search in the geometrical tree (Fig. 3). Instead of a linear or binary search (time spent proportional or logarithmic with the number of volumes, respectively), a 'direct access' technique has been developed to make the time basically independent of the number of volumes. Every volume containing other volumes (node) is 'virtually' divided into equal slices at initialization time (the best direction is computed automatically by the program according to statistical considerations and the number of volumes found per non-empty slice); for each slice, a data structure is filled with a list of the volume IDs intersecting such slices; slices with identical lists point to the same contents and are merged. This is possible only because the relation of order between the slices exists already at initialization time: it is not determined by the direction of the particle, but by the chosen axis. Therefore, not only is it possible to collect and merge empty cells, but even any set of cells with identical contents. At tracking time, it is straightforward to find in which slice the particle is and only its contents have to be checked. The same is true for finding the next boundary to be crossed: only if the intersection point with a content lies outside the current collection of slices, will the next one be checked. Therefore, we can afford the maximum granularity to find the current location of the particle and the minimal needed granularity to find the next boundary.

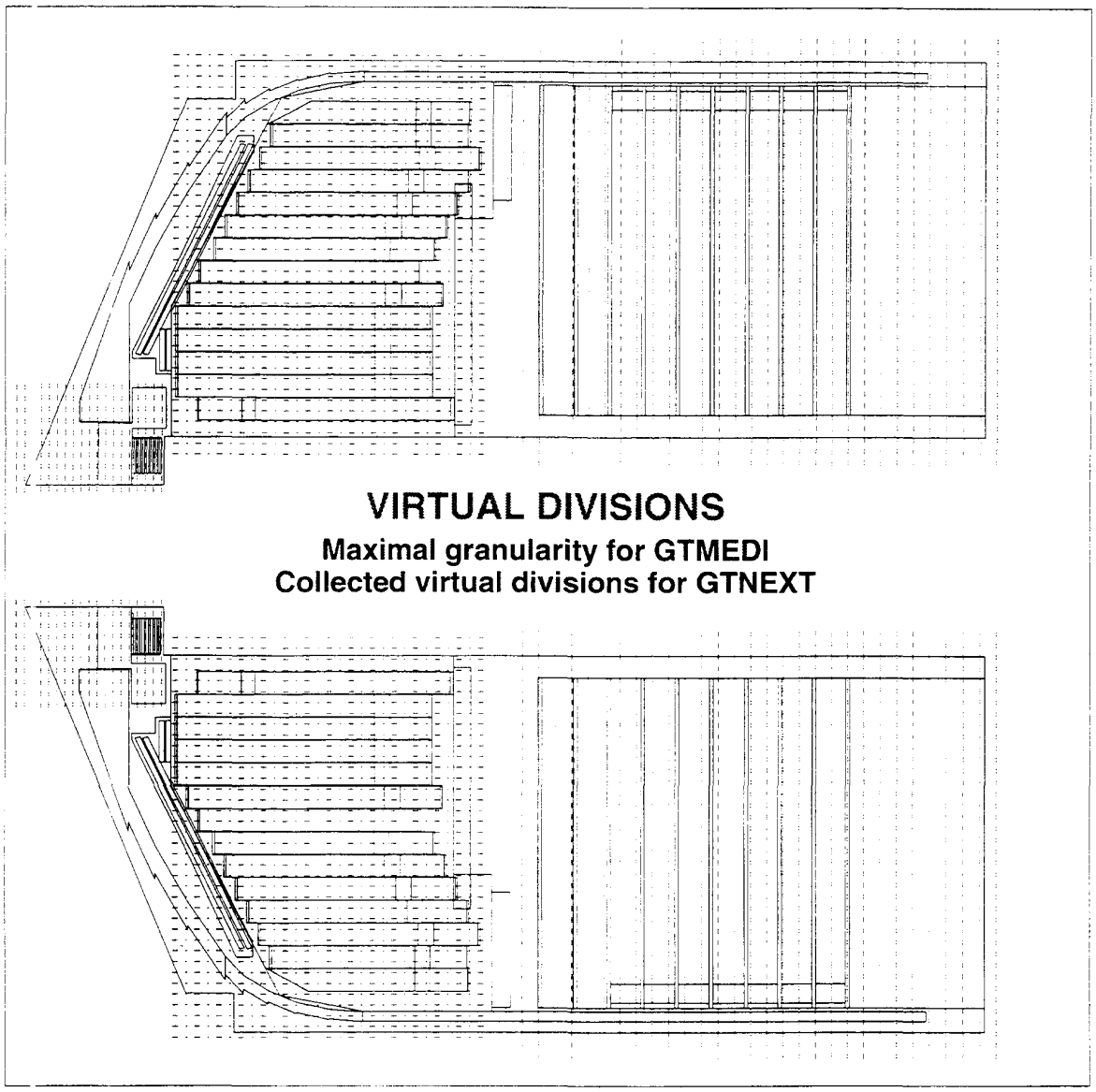

Figure 3

Finally, it is straightforward to divide the 3-D space with proper grids by overlapping bounding volumes 'virtually divided' along different axis (Fig.4).

The new algorithm gives on average about a factor two in speed for the overall simulation in the LHC and LEP detectors (Fig. 5). It also allows a fast tracking even in geometrical structures received from CAD systems. The initialization time for a detector like ATLAS (11 million volumes) is less than 1s on an HP735 and the size of the specialized data structure for the tracking is about 50000 words.

4 GEANT and PAW classes

GEANT uses the CERN library package ZEBRA to generate data structures to store different kinds of data. GEANT being a HEP simulation package for GEometry ANd Tracking, these data are:

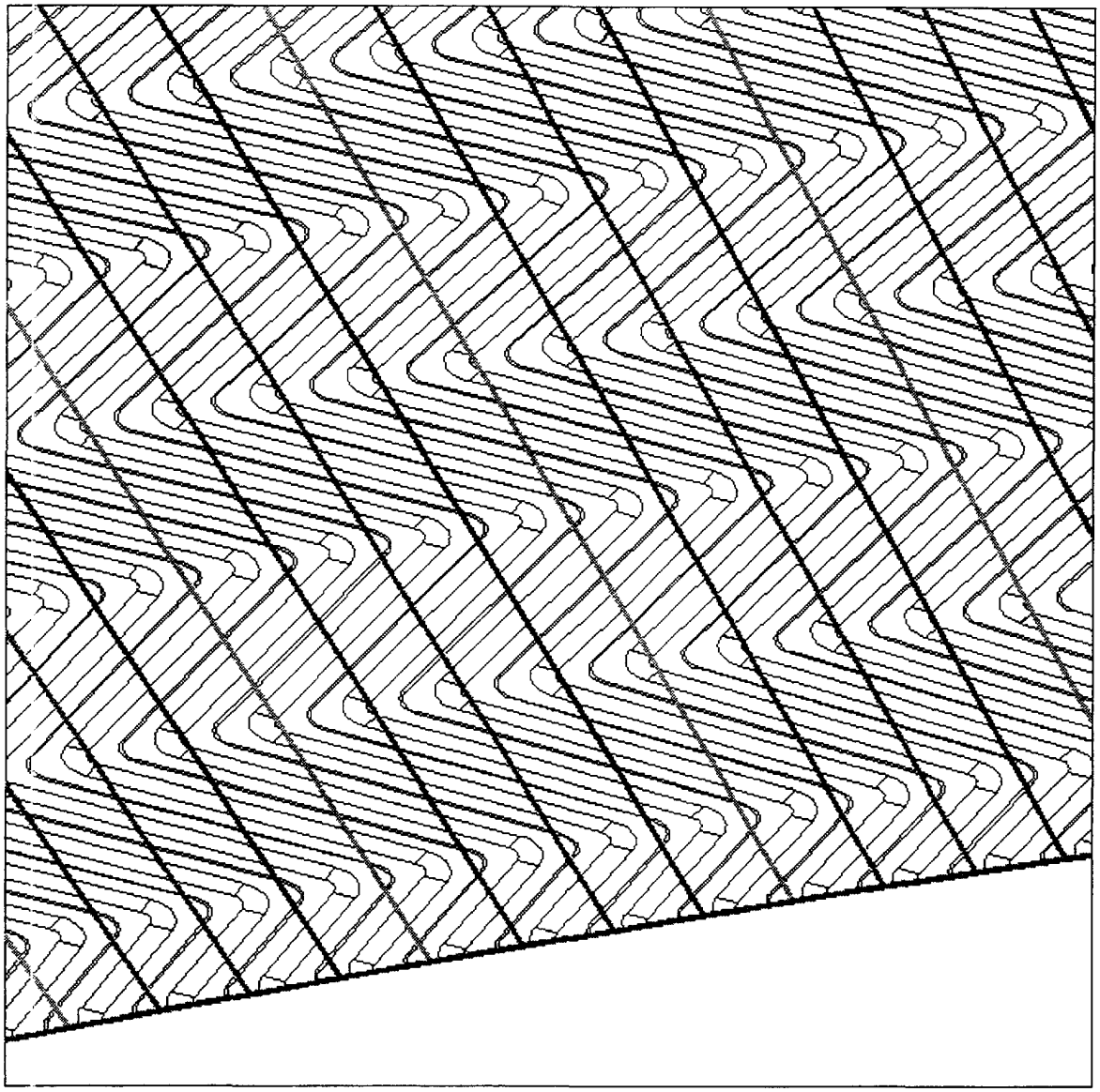

Figure 4

- volumes and their interrelationship,
- particles and their properties,
- materials and the cross sections of various particles,
- vertices of particles,
- tracks of particles,
- hits in various detector parts,
- digitized data of the hit structure.

The OSF/Motif based version of GEANT, realized using the KUIP/Motif interface provides the user with an interactive, direct access to the data structure. We define the GEANT and also the PAW classes as browsable classes, by which we mean the data structures inside the application. The application (GEANT/PAW) itself defines the functionalities one can perform on the data structures. We call a browsable object the instances inside a class.

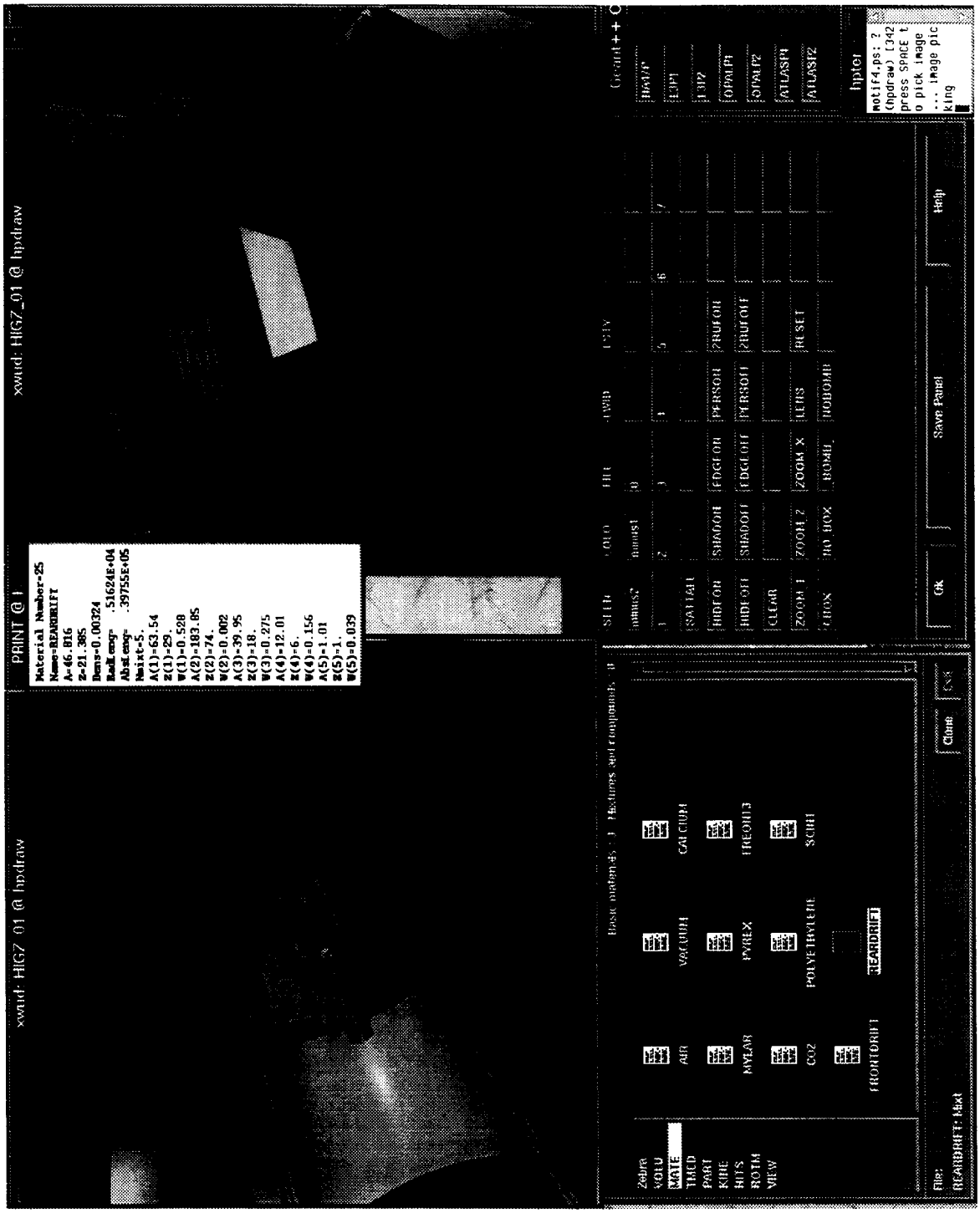

Figure 5

4.1 KUIP, a user interface management system

As mentioned in the section above, the interactive user interfaces are realized using KUIP, which stands for Kit for a User Interface Package, and has been developed at CERN in the context of GEANT and PAW. It is a User Interface Management System (UIMS) which is

a software toolkit intended to provide a homogeneous environment in which users interact with different kinds of applications, and also to provide some development tools to help application developers in making an application program interactive.

A good UIMS design should achieve the best compromise between the ease of use (e.g. including a maximum of online-help requiring a minimum of additional written documentation) and the avoidance of frustration of experienced users (e.g. not restricting only to graphics menus and not insisting on deep menu hierarchies). The heterogeneous user base of GEANT and PAW at different levels of experience requires a multi-modal dialogue, i.e. different dialogue styles with the possibility to switch between them inside the application. A beginner or casual user may prefer a menu mode for guiding him through the set of commands, while the user who is already familiar with an application can work more efficiently with the command line mode. This leads to the mixed control facility, i.e. the possibility to have either the command processor (master) controlling the application (slave) or vice-versa. In the first case the command processor prompts the user for the next command and passes it to the application. In the second case the application program itself can decide to give control to the user (through the command processor) or to ask directly the command processor to execute a command sequence. The two CERN library packages (GEANT/PAW) combine a common concept of browsable classes. This concept provides the two packages with a similar seamless interface.

4.2 The layers of KUIP

As a User Interface system KUIP concerns both the application writer and the application user. Fig. 6 shows the different layers in a KUIP-based application. The ordinary user only has to know about the top layer with standard rules how to select a command and how to supply the necessary arguments. The application writer on the other hand has to understand the tools which allow to define the command structure for the middle layer, and he has to provide the bottom layer which implements the specific code for each command.

4.3 The concept behind

4.3.1 The application writers's view

The application writer has to describe the command tree structure and the parameters and action routines associated with each command. In order to do this job KUIP has to store this information in internal structures. The application writer also has to provide the middle and bottom layer (application and action routines). In order to provide its functionality KUIP has to store this in internal structures. The possibility that the application programmer should himself write the routine calling the appropriate KUIP routines was considered to be too inconvenient. Instead the application writer only has to provide a text file called the Command Definition File (CDF) containing a formal description of the command set. A special program, the KUIP Compiler (KUIPC) analyzes the CDF and generates a file containing the source code (i.e. calls to KUIP routines) to store the command structure at run-time. Generating the actual command definition code automatically from the higher-level CDF format offers many advantages:

- the directives needed to mark-up the CDF description are easier to learn than the calling sequences and the correct ordering of the definition routines,
- the command structure is more visible in the CDF than in the corresponding source code cluttered by calls to cryptic routine names with cryptic option arguments.

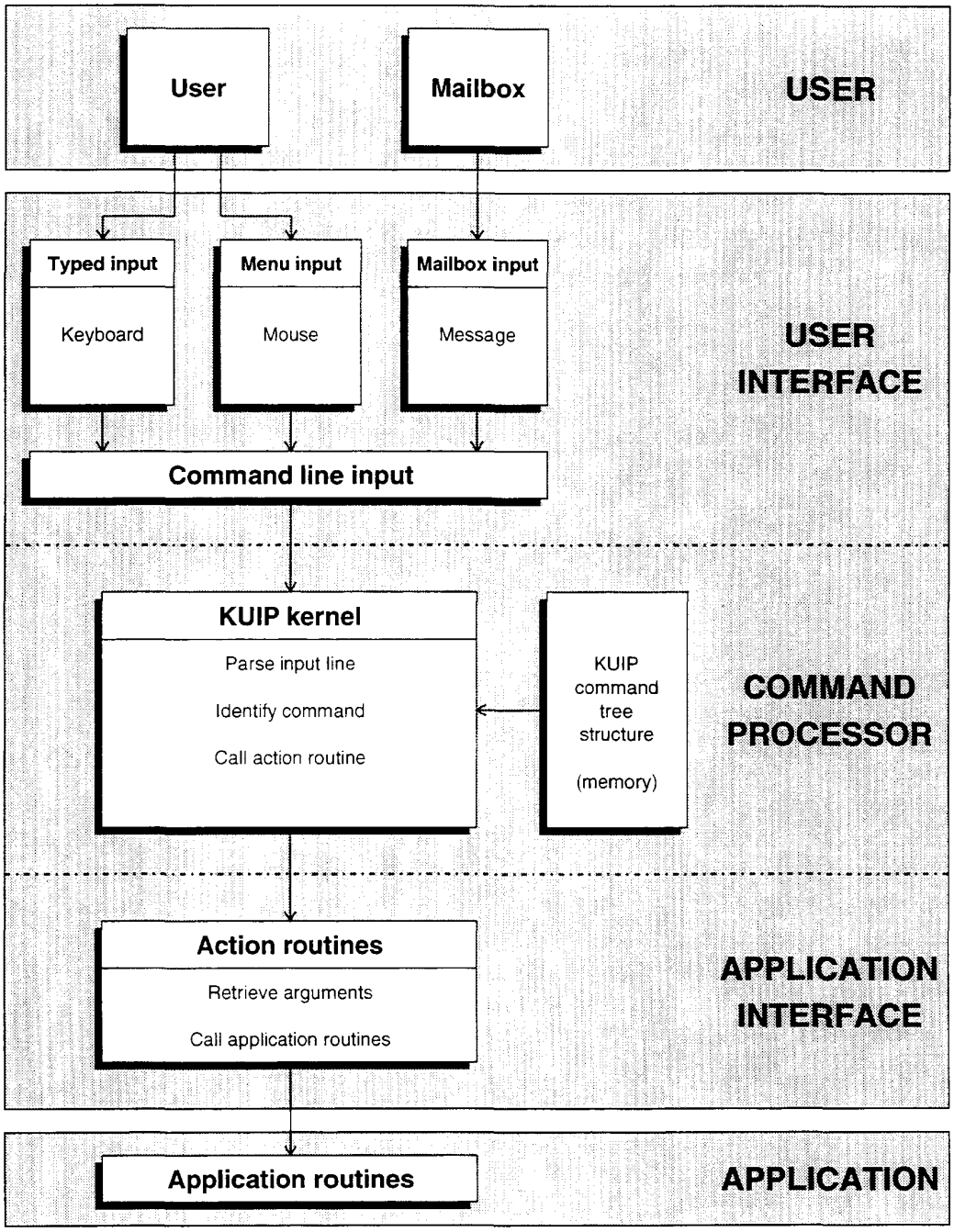
User
Mailbox
USER
Typed input
Keyboard
Menu input
Mouse
Mailbox input
Message
USER INTERFACE
Command line input
KUIP kernel
Parse input line
Identify command
Call action routine
KUIP command tree structure
(memory)
COMMAND PROCESSOR
Action routines
Retrieve arguments
Call application routines
APPLICATION INTERFACE
Application routines
APPLICATION

Figure 6

4.3.2 The application user's view

KUIP provides different dialogue modes for how the user can enter commands:

- the default command line input from the keyboard,
- various menu modes either driven by keyboard input or by mouse clicks. Switching between dialogue styles is possible at any moment during the interactive session.

5 GEANT4: DRDC proposal for an object-oriented simulation toolkit

The DRDC proposal P58 is discussed. A full list of the authors can be found in [7].

5.1 The motivation and the purpose of the project

For the Large Hadron Collider (LHC) and heavy ion experiments, an even larger degree of functionality and flexibility is required in GEANT, thus making necessary a re-design of the program. It is intended to investigate the use of the promising Object-Oriented (OO) techniques to enable us to meet these goals.

The philosophy should no longer be 'GEANT simulates all the known physical processes at all the energies', but it should rather be 'the simulation of any physical process at any energy can be easily inserted into GEANT by any user'.

The functionality needed for the LHC experiments requires that users be more easily able to modify general purpose code to suit their specific needs. Other diverse requirements, again pointing to the need for a flexible toolkit, come from heavy ions physics (multiplicities per event) and from CP violation and neutrino physics (very large statistics).

With an object-oriented programming style one aims to exploit:

- Data encapsulation, to provide better software maintainability. The data part of an 'object' can only be modified by methods listed in its 'class' definition. This dramatically limits the amount of code that has to be examined in case of problems, compared for example, to the traditional approach, where any procedure can modify data in a Fortran COMMON block it contains. Such information hiding will also allow the optimization of the internal representation of an object without affecting its use.
- Inheritance (and multiple inheritance), to improve software productivity. New classes can be derived from existing ones, inheriting all of their features — only the specific features of the derived classes need to be implemented.
- Polymorphism, to permit easier extension of the system. For example, the different geometrical shapes could be classes derived from a common base class. In order to introduce a new shape, only the shape dependent 'methods' have to be written. The name of the methods can be the same as in the base class and even the same for all the derived sub-classes. The code in which the objects are used does not need to be modified — in complete contrast with conventional programming languages.

It is therefore proposed to achieve the goal by re-designing GEANT according to the object-oriented paradigm (to be implemented in C++), in order to transform it into a general toolkit where the user can easily plug in code to extend the functionality of the program. For example, it must be possible to add a new geometrical shape to the GEANT modeller, add new physical processes, to introduce cross sections for the tracking of an ion fragment in a given material, or to 'overload' a library routine in as simple a manner as possible. Pioneering work in this field has been done in the context of the GISMO project [4].

In addition, GEANT is increasingly requested for applications such as tomography, dosimetry, space science, etc., and the above enhancements could also be used to advantage by people working in those fields.

5.2 The Development

5.2.1 The data model

The present data model and all the data structures in GEANT have been implemented using the ZEBRA package [5]. A very efficient and powerful data model is required for the geometrical modeller, because several million volumes need to be defined in order to simulate an LHC detector and it must be possible to rapidly navigate in such a big data structure at tracking time. With the current version of GEANT (3.21) it is indeed possible to achieve an effective simulation of the LHC detectors at their present status of design; however, we still need to improve the program since the complexity of such detectors will increase even further.

Today a typical experiment uses several different data models for its on-line, reconstruction and simulation software, which limits the possibility of an effective data exchange between the different programs. Our goal for LHC should be to proceed towards a unified data model (actually the words 'a common object model' are more appropriate in a OO framework).

To reach an object-oriented design in the new simulation tool, the way data are stored and inter-related has to be thought out in detail. Studies have already been carried out to explore how the existing data structures can be described in an object-oriented model. In this exercise the Object-Modeling Technique (OMT) methodology has been followed and an appropriate CASE tool has been used by the OO group from KEK, in Japan.

Deeper investigations have to be foreseen to ensure the maximal functionality, flexibility and run-time performance of the new simulation program. This will probably imply modifications to the object model.

5.2.2 The kernel

The GEANT kernel consists of two major parts: one part handles the geometrical description of the detectors, and the other tracks particles through the defined geometry.

A serious drawback of the present procedural approach in GEANT is the difficulty of extending functionality. When one wishes to implement an additional geometrical shape, nearly 60 subroutines must be modified. The same is true if one wishes to define a new particle type to be tracked. In addition, in the current GEANT, limitations come from precision problems in the tracking and rotation matrices.

The OO GEANT kernel will still consist of the same two major components: geometrical modeller and tracking system. The re-design of the kernel will give us the opportunity to revise several algorithms, which provoked the previously stated problems. The following considerations will be kept in mind:

- With an OO-based geometry and tracking, via the two mechanisms of inheritance and polymorphism, every GEANT class can be extended and methods can be overloaded in a transparent way.
- An important challenge facing this project is finding an optimal use, in the framework of OO development, of the experience acquired from the logical and algorithmical problems solved in the past. Close attention will be paid to performance aspect.
- Some algorithms need to be revised: for example, the code to perform tracking in a magnetic field (conceptually delicate because of its connection to multiple scattering and to energy loss) has to be re-written in a more robust and efficient way. Moreover, a careful analysis of the actions to be taken when particles reach volume boundaries,

together with the use of appropriate precision in all of the geometry classes, should help to resolve precision problems once and for all.

- The possibility to provide a FORTRAN 77/90 [6] (and/or RZ [5]) callable interface to the new geometry should be investigated: this would permit testing of the new GEANT on existing detectors at the Large Electron-Positron Collider (LEP) and others, using GEANT 3.21 as a reference to compare physical results, maintainability, extensibility and performance.
- The re-design of the geometrical modeller will provide an opportunity to make it more compatible with CAD systems (possibly via the STEP interface). In particular, techniques for tracking in BREP (Boundary REPresented) solids should be investigated.

5.2.3 The User Interface

At the present the user has several interfaces to the GEANT program:

- the 'batch customization' interfaces, which are used to build executables for massive batch production of simulated events:
  - User routines, where the user can put his/her own code at the level of intialization (UGINIT), run (GUTREV), event creation (GUKINE), tracking (GUTRAK, GUSTEP) and termination (GUDIGI). The same is true for the definition of the geometry via the user routine UGEOM (calling the library routines GSVOLU, GSPOS, GSDVN).
  - 'Data records', to turn on and off several tracking options.
- the 'human interactive' interfaces, which have been developed to enhance the functionality of GEANT as a detector development and particle physics education tool:
  - A classic command line interface to set the values of the data records or call GEANT subroutines. This uses X11 and PostScript (via the HIGZ [8] interface) for graphics.
  - A KUIP-Motif-based user interface which contains the same functionality as the X11 interface version, but provides a 'KUIP object' browser by which the user can access the GEANT data structure.

The planned object-oriented simulation tool will be designed to be interactive from the outset. The user would then automatically benefit from modern system environments, including features such as shared libraries and dynamic linking. As a result, the ever-increasing size of the executable modules could be better controlled. In addition, the user would benefit from a better development environment to study the detector design and response.

5.2.4 The physics

GEANT provides a detailed description of the various physical interactions that elementary particles can undergo in matter. This list of processes is well-integrated in GEANT for electromagnetic and muonic interactions. Hadronic interactions are handled via interfaces to external programs (GHEISHA [9], FLUKA [10] and MICAP [11]).

There is a large asymmetry in the internal data structure describing electromagnetic and hadronic interactions. This is an impediment to a coherent treatment of particle interactions.

Another limitation concerns the adjustment of various interactions to match experimental data. This is relatively simple in the case of electromagnetic interactions, where the relevant code is well integrated in GEANT. In the case of hadronic interactions, however, the situation is rather complicated and the user must invest a significant amount of time if he needs to understand the internal logic of the external packages. In addi-

tion, the introduction of a new interaction process requires the modification of about 20 subroutines.

The new OO simulation tool should provide the same physical processes as currently provided by GEANT. An object-oriented design should result in more convenient inclusion of new processes or modification of existing processes by the user.

The GEANT team has developed a close relationship with experts in the description of electromagnetic and hadronic physics. Their experience will be an essential contribution to the OO design of GEANT, where the description of hadronic interactions should be well integrated in the program, as is the case for electromagnetic interactions.

In previous versions of GEANT a data structure of cross sections was set up only for electromagnetic particles; this data structure was used as a lookup table for electromagnetic interactions. The cross sections for hadronic interactions were however calculated at each tracking step. This approach of a lookup table could also be applied to hadronic interactions — at least for particles with a long lifetime. This would lead to a generalized handling of electromagnetic and hadronic cross sections and interactions, which will in itself allow the generalization of the tracking of hadrons and electromagnetic particles.

Users have frequently requested the possibility to be able to plug-in their own fast parameterization of physics processes and particle cascades in various detector parts. This request will be taken into account at the design level since something much more powerful and flexible than what is available in the current versions is needed for LHC.

5.3 Relation to existing libraries

5.3.1 CERNLIB

Compatibility with existing CERNLIB packages (such as RZ, HBOOK [12], KUIP), which today provide essential functionality for GEANT, is initially required in order to test OO GEANT, without needing to rewrite all of the underlying software layers. However, we would foresee that these packages would probably be upgraded to OO over a period of time. Indeed, for RZ there is a specific proposal [13] to do this.

5.3.2 HEP Class libraries

Existing class libraries, such as CLHEP [14], have to be examined in order to investigate if it is possible to use their basic functions for our purpose, such as point/vector operations, particles and tracks definitions, linked-list management, garbage collection, tree search, look-up tables, etc. A collaboration with other R&D projects related to OO will be essential to ensure a common framework by the use of the same class libraries.

5.4 Milestones

The project will be a collaboration between researchers working in the experiments, the GEANT team, and simulation experts. From the software engineering point of view, the project will follow the 'spiral model' of software development: therefore the analysis, design and implementation phases of the various modular parts of the program will be iterated over the duration of the project.

There are two factors to be considered concerning the timescale: one is the development time of the new program and the second is the support of the current version of GEANT (3.21): it is essential to remember that a production tool must still be available to the experiments taking data or being simulated today (LEP, HERA, LHC itself). The following milestones are proposed:

- Phase 1: first iteration of analysis-design-implementation stages completed and prototype available in the first year of work (by the end of 1995); this work will be

performed in close collaboration with RD41 [15] and the proposed P59 [13] project to proceed towards a unified data model for simulation, reconstruction and I/O. In particular, we aim to:

- have a complete class design for the new simulation program, which includes geometry, tracking, physical processes, kinematics, digitisation, etc.,
- a first implementation of the principal 'methods' for each class,
- a working prototype for geometry and tracking based on such a design.

- Phase 2: working version of the program by the end of 1996; this includes:
  - an evolution of the first model, based on the requirements and the feedback coming from the results of tests and prototyping in Phase 1,
  - the tracking system should be finalized,
  - the geometry should be developed to achieve the current functionality of GEANT 3.21,
  - the physics should be structured in an object-oriented way, although not yet rewritten in an OO language.

  In this stage, the program can also be used by the current experiments via the standard FORTRAN/RZ geometry interface. It can be compared to the current GEANT 3.21. Above all, it can be already used as an OO toolkit and its functionality can be extended by a C++ programmer.
- Phase 3: final version of the program by the end of 1998; this requires:
  - completion of the development of the geometrical modeller and data base compatible with CAD systems,
  - completion of the development of the physics processes,
  - completion of the development of a common framework and user interface with the reconstruction programs, I/O and event generation libraries,
  - a systematic testing of the program by the experiments to refine and optimize both design and algorithms.

6 Conclusions

The GEANT tracking allows the visualization of detectors composed of millions of objects. A new way of showing simulated events has been presented, in which time steps are driving the visualization, allowing a better understanding of the underlying physical processes. In addition, new tracking algorithms have been introduced providing a powerful tool to navigate in extremely complex geometrical structures. The proposal of object-oriented re-design of GEANT has been presented and discussed in detail.

7 Acknowledgements

The GEANT team would like to thank D.Boileau from the Directorate Services Unit for his help in the realization of the video and F.Rademakers for his help in setting up the hardware equipment for the practical exercises. Further thanks to HP Suisse in providing us with the hardware equipment for the CERN School of Computing.

References

[1] GEANT User Guide, CN/ASD, CERN, 1994.

[2] CN/ASD, 'KUIP', CERN, 1994.

[3] PAW User Guide, CN/ASD, CERN, 1994.

[4] W.B.Atwood et al., 'The Gismo Project', C++ Report, 1993.

[5] R.Brun et al., 'ZEBRA, User manual', 1988.

[6] M.Metcalf and J.Reid, 'Fortran 90 explained', Oxford, 1992.

[7] GEANT4: An Object Oriented toolkit for simulation in HEP, CERN/DRDC/94-29, 1994.

[8] CN/ASD, 'HIGZ', CERN, 1994.

[9] H.Fesefeldt, 'GHEISHA', Aachen, 1985.

[10] T.Aarnio et al., 'FLUKA user's guide', TIS-RP-190 CERN, 1987,1990.

[11] J.O.Johnson and T.A.Gabriel, 'A user's guide to MICAP', ORNL, 1988.

[12] CN/ASD, 'HBOOK', CERN, 1994.

[13] DRDC/P59, 'A persistant object manager for HEP', CERN/DRDC/94-30, 1994.

[14] L.Lonnblad, 'CLHEP, a class library for HEP', CERN, 1992

[15] DRDC/P55, 'MOOSE, Methodologies...', CERN/DRDC, 1994.