VISUALISATION IN HIGH ENERGY PHYSICS

L. Taylor
Department of Physics, Northeastern University,
360 Huntington Avenue, Boston, MA 02115, USA

Abstract

We describe the concepts underlying successful HEP visualisation systems, for both statistical analyses of multi-event data sets and for the display of individual events in the detectors, using examples from current experiments. Future HEP experiments, such as those at the CERN Large Hadron Collider, are considerably more complex than those currently running. We discuss how new computing technologies will facilitate the difficult visualisation tasks of these experiments.

1 INTRODUCTION

Visualisation plays a crucial role in enabling physicists to understand complex multi-dimensional data. With the advent of powerful computing hardware, sophisticated scientific visualisation software has become a standard part of the analysis toolkit of High Energy Physics experiments. Fig. 1 shows schematically the flow of data, represented by arrows, between the traditional software modules used for HEP offline analysis, represented by rounded boxes. The box labelled "Physics analysis" contains many graphics tools which are not specific to a particular experiment, as well as dedicated ones such as event visualisation programs. In this paper emphasis is placed on the HEP-specific graphics tools in use. No attempt is made to review the multitude of other graphics applications such as text editors, document preparation systems, visual programming tools, WWW browsers, collaborative working tools such as teleconferencing [1], Computer Aided Design programs, and so on. In particular, after a brief overview of general principles, we describe in the subsequent sections the tools which are most commonly used for the statistical and numerical analysis of multi-event data sets and for the visualisation and analysis of single events. The package predominantly in use for the former is PAW [2] while the latter includes the generic GEANT package [3–5] and many experiment-specific event display programs.

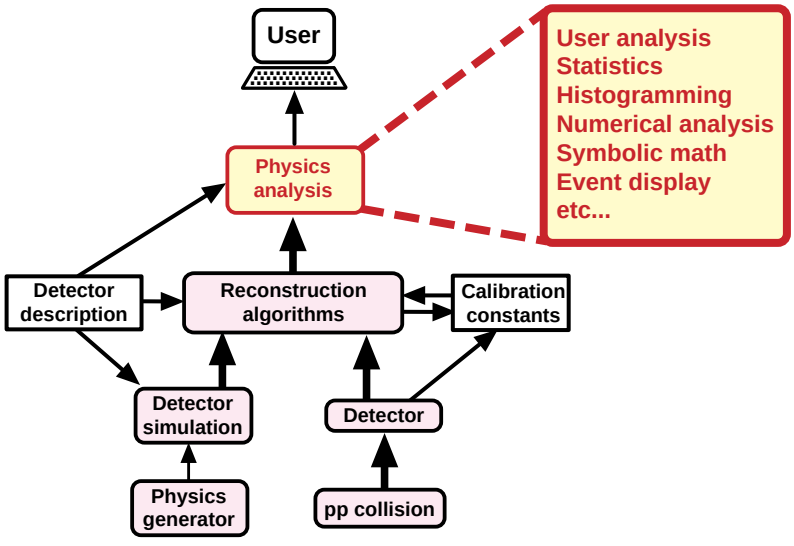

Fig. 1: Schematic flow of data between the main software tasks of a typical physics analysis system.

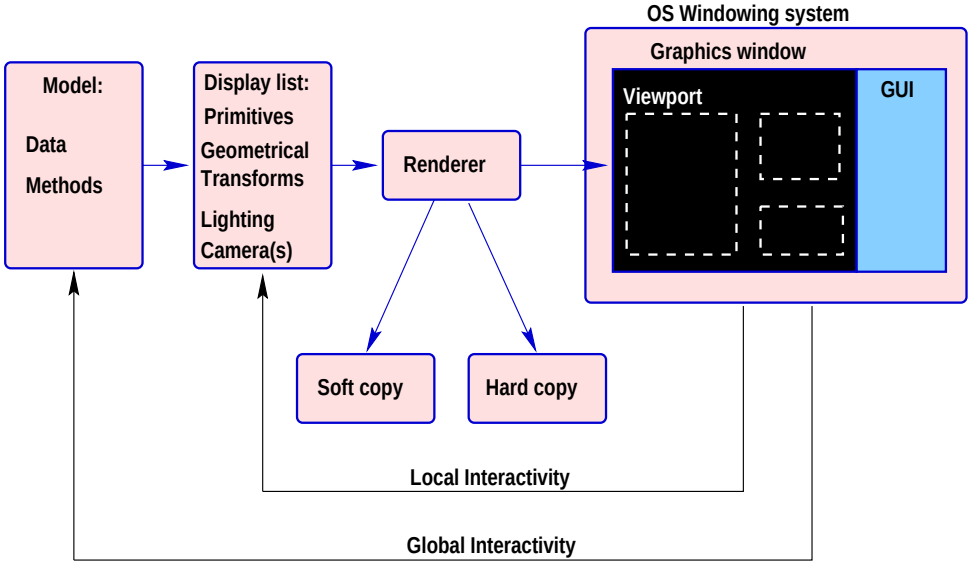

Fig. 2: The basic components of a generic graphics system.

2 GENERAL PRINCIPLES

The basic concepts of computer graphics systems are fairly general, although the implementation, efficiency and underlying technologies of specific packages can vary considerably. In the following sections we describe each of the generic components of a graphics system, shown schematically in Fig. 2.

2.1 Graphics Model

The graphics *model* is the underlying system which is to be visualised. In general it is non-graphical in nature. In HEP, the model consists of: equations, coded algorithms, data, with no associated geometrical description; descriptions of physical entities (e.g. HEP detector) with associated geometry and other properties (radiation length, temperature, stress tensor, etc.); abstract entities (e.g. HEP events) with associated geometry and other properties (specific ionisation, energy deposited, etc.); multi-event data sets; and physics event reconstruction algorithms. The model may contain both data and methods.

2.2 Graphics Primitives

The fundamental graphics objects to be drawn are known as *primitives*. These include polymarkers, polylines, filled areas, arcs, text, splines, surfaces, and so on. The basic primitives are provided by the underlying graphics library and are optimised in conjunction with hardware. In addition the developer may define *super-primitives* which use one or more of the basic primitives (for example in a class library sitting on top of the basic library) such as a silicon wafer with strips or a particle trajectory with the associated detector hits.

2.3 Geometrical Transformations

Having built up a graphical entity from one or more primitives, its geometrical specification (offset, scale, and orientation) in space must be defined. It may be convenient to define the geometry in terms of either a fixed reference frame or relative to another object. Moreover, the geometrical specification of the graphical entities may change with time, for example as the user changes the scene or the point of view. Consider the following geometrical transformations[1]

[1]We describe the 2-D transformations for simplicity. It is straightforward, if tedious, to extend the treatment to 3-D.

Translation:
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \delta_x \\ \delta_y \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix} \qquad P' = T + P$$

Scaling:
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \qquad P' = S \cdot P$$

Rotation about (0,0):
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \qquad P' = R \cdot P$$

For conciseness and efficiency we would prefer to have a uniform treatment of the three types of transformation and an algebra enabling them to be combined into a single operation which can then applied to the many graphical objects. We therefore introduce a *homogeneous coordinate* system which requires the addition of an extra (non-physical) dimension, as follows:

Translation:
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{pmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad P' = T \cdot P$$

Scaling:
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad P' = S \cdot P$$

Rotation:
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \qquad P' = R \cdot P$$

Using homogeneous coordinates, all operations consist of matrix multiplication. Thus, an arbitrary sequence of translations, scales, and rotations can be described by a single matrix. Such a transformation constitutes an *affine* transformation (preserving parallelism of lines but not lengths and angles). It is therefore important to apply the transformations in well-defined order. For example, Fig. 3 shows the operations required to rotate an image of a house about an arbitrary point (a, b). The net *single* transformation, to be applied to all points of the house, is:

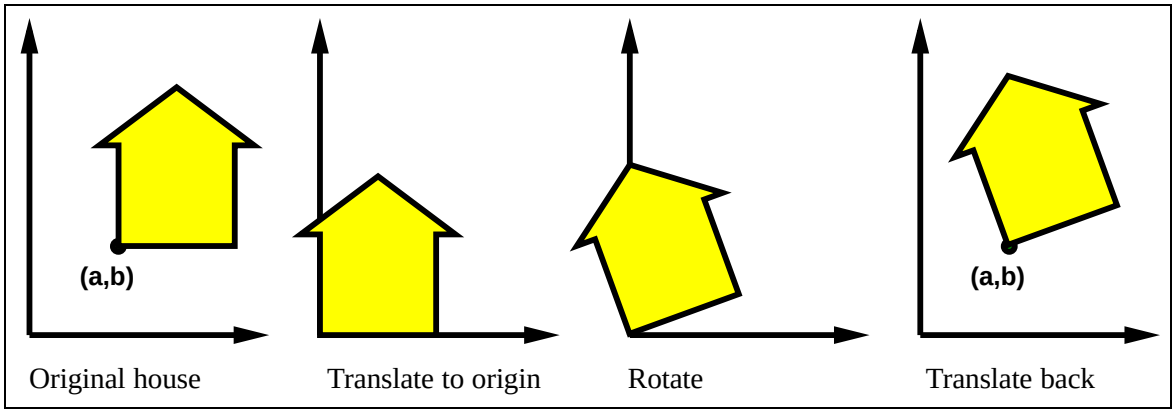

Fig. 3: Rotation of an image about a point (a, b).

$$T(a,b)\cdot R(\theta)\cdot T(-a,-b) = \begin{pmatrix}1 & 0 & a\\ 0 & 1 & b\\ 0 & 0 & 1\end{pmatrix}\cdot\begin{pmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{pmatrix}\cdot\begin{pmatrix}1 & 0 & -a\\ 0 & 1 & -b\\ 0 & 0 & 1\end{pmatrix}$$
$$= \begin{pmatrix}\cos\theta & -\sin\theta & a(1-\cos\theta)+b\sin\theta\\ \sin\theta & \cos\theta & b(1-\cos\theta)-a\sin\theta\\ 0 & 0 & 1\end{pmatrix}$$

2.4 Rendering

Prior to rendering (or drawing) an image, the desired graphical representation of the model must be specified in terms of graphics primitives with attributes (colour, opacity, etc.), their locations and orientations in space, and a lighting model, described in terms of ambient, planar, spot, and directional light sources. For reasons of performance, this description is often stored internally by the graphics package as a *display list*. The rays are then traced through the scene and reflected according to the colour and direction of the light and the geometry and optical characteristics of the surfaces of the graphics primitives. The rays are traced, possibly through occluding media, to form an image on the virtual viewing plane of the observer (sometimes referred to as the camera). The light reaching this plane from many sources is combined to form the final image which is then reproduced on, for example, the cathode ray tube (CRT) of a computer screen.

The rendering process must be highly optimised in order to obtain satisfactory performance in particular when dealing with many of the complications not described here [6], such as reflection (diffuse and specular), refraction, transparency, shadows, depth cueing, texture and bump mapping, anti-aliasing, hidden-line and surface removal, and colour mixing. Genuine 3-D images using stereo vision requires two rendering operations for the cameras associated to each eye.

2.5 Interactivity

In general, the user wishes to interact with the graphical display system. Such interaction may be classified as either *local* or *global*, as shown schematically in Fig. 4, which follows the notation used by Terry Hewitt [7]. Local interactivity consists of operations which affect the view but not the representation of the model, such as the rotation, translation, or zooming of the camera which defines the viewpoint of the observer. The changing of the visibility of objects in a scene is also a locally interactive operation. For reasons of performance, locally interactive operations are generally performed on the display list, without recourse to the underlying model.

Global interactivity refers to operations which involve the underlying model. Examples include the addition to the scene of objects which are not currently on the display list, a non-Euclidean coordinate transformation of a scene, or the updating of an existing graphical object to reflect a change in the status of the corresponding object(s) of the underlying model.

2.6 Graphical User Interface

The Graphical User Interface (GUI) of a visualisation program relies heavily on graphics, albeit with a somewhat different emphasis and distinct software tools to those used for the graphics application itself. The GUI should enhance the usability of the application by minimising the learning time, reducing errors, and increasing the efficiency with which complicated command sequences can be executed. A well thought out conceptual design for the GUI is crucial in ensuring the ease with which the application can be used.

The GUI should be self-consistent and resemble, wherever possible, other applications with which the user is already familiar. Visual consistency would imply, for example, that toggle buttons are square,

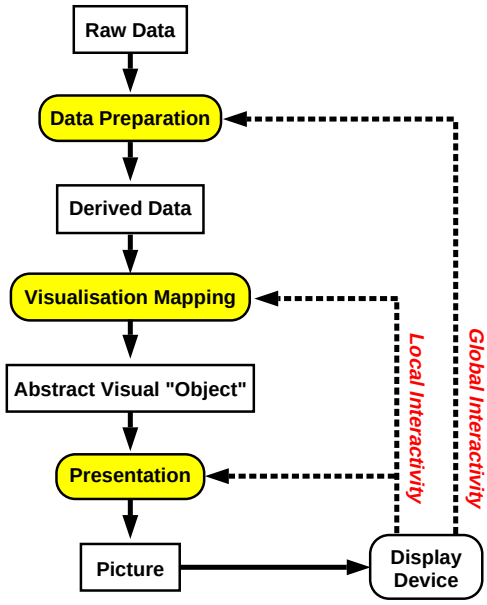

Fig. 4: The rendering pipeline showing local and global interactivity.

radio buttons are diamond, the default button which is activated by the `RETURN` key should have a heavy outline, and so on. Generic commands, such as copy, cut, paste, delete, or move, should perform the intuitively expected operation independent of whether the object is text, a graphical object, etc. The GUI should conform to the *Law of Least Surprise* in which the result of an action matches the expectation of the user.

The GUI should not allow a user to do inadvertent damage, either to data or the execution of the program, as a result of misunderstanding the GUI. For example, potentially serious operations such as deleting a file or data might invoke a dialogue box requesting confirmation prior to completing the command. The design of the GUI should be such that the user is free to learn by experiment, by inclusion of error recovery mechanisms such as the `Abort`, `Cancel`, and `Correct` operations. Ideally, the GUI should also include a multi-command `Undo` feature, with a corresponding `Redo` feature, to permit the user to reverse the effects of an undesired action.

The GUI should support multiple levels of expertise. For example, a beginner should be able to find all desired actions graphically (for example by using just the mouse) while the expert should be permitted to use keyboard accelerator sequences for common operations such as copy, paste, or delete. Similarly the mouse itself can be overloaded for expert use with, for example, distinct commands assigned to single and double clicking. It may prove useful to map each operation of the GUI to a textual command prior to its execution, thereby enabling the expert user to enter commands as text, to execute macros containing many commands, to log the actions of a complete session, and so on.

While a well-designed GUI should be largely self-explanatory to the user, even the simplest application program should also incorporate a help facility. This should ideally include full explanations as well as concise context-sensitive help. A familiar example is the *bubble-help* method in which a small "bubble" containing textual help appears, after a small time delay, as the mouse cursor moves over a particular element of the GUI.

2.7 Graphics Software

The core offline software of current HEP experiments, shown schematically in Fig. 1, is typically written in Fortran 77 and C, with some non-standard extensions. The data are managed and stored using a non-

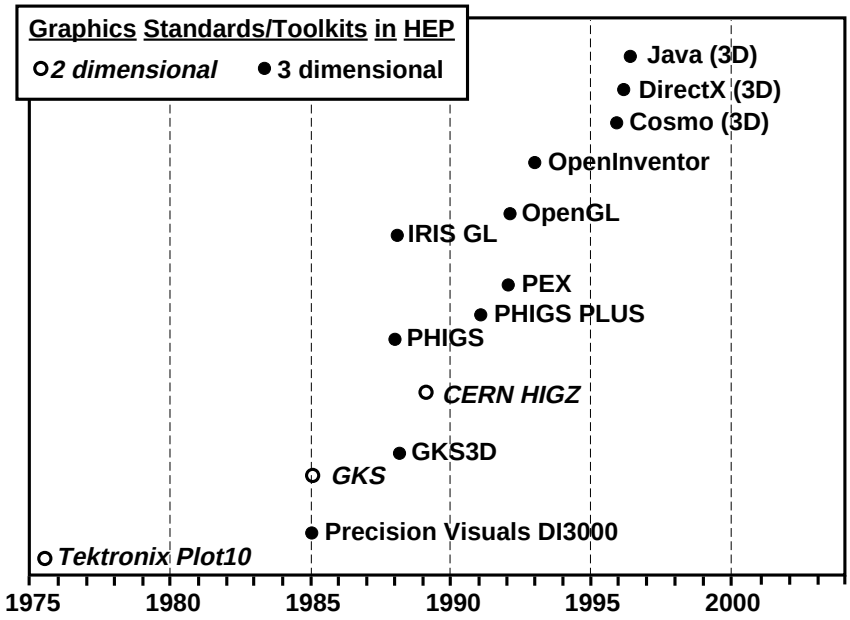

Fig. 5: Standard graphics libraries used by the High Energy Physics community.

commercial package such as ZEBRA [8] or BOS [9]. Code management usually relies on PATCHY [10], CMZ [11], or cvs [12]. The code development process is evolutionary (Darwinian not e.g. Booch'ian!) in that many physicists contribute code, some of which is successful and survives while other parts die out naturally. The resulting code typically has considerable functionality but little semblance of design; it is therefore rather difficult to maintain and subject to bugs.

The use of standards greatly facilitates the software development process and the maintenance of the code which is developed and used by many hundreds of physicists running on multiple platforms and architectures linked by local- and wide-area networks. Standards are provided by national agencies (ISO, ANSI, BSI, DIN, etc.), professional organisations (IEEE, X Consortium, etc.), or in a proprietary, open, or de-facto manner by commercial companies. Fig. 5 shows the standard graphics libraries most commonly used in HEP over the past few decades. Currently, the most prevalent standards used are GKS [13, 14] and PHIGS [15]. The use of GKS has diminished greatly in recent years with the advent of PHIGS, which is in turn starting to be replaced by the OpenGL [16] graphics library. With the increasing use of an object-oriented paradigm as discussed in section 5 the use of OpenInventor [17], which is a higher-level class library based on OpenGL, is increasing.

Standard GUI environments, or windowing systems,include: X Windows (open); Windows 95 and NT (Microsoft); SunView (Sun); and Macintosh (Apple). HEP graphics applications are typically based on X Windows [18], running on Unix platforms. Widget sets include the Athena widget set (X Windows), Motif (OSF collaboration of HP, Digital, Microsoft, IBM, etc.), OpenLook (Sun and AT&T), and Tcl/Tk. The widget set most prevalent in HEP is Motif [19].

Many image formats are used to greater or lesser degrees in HEP, including: GIF, JPEG, TIFF, BMP, etc. for static images; MPEG, Quicktime, AVI for multiple images; and Postscript, CGM, IV, VRML for graphics metafiles (soft copy). The support for a standard hard copy format is important since it is portable, high resolution, free of format problems, well-suited to long-term archival storage, and facilitates rapid viewing even for very large documents. The preferred standard in HEP for producing graphical hard copy is Postscript [20]. In general, vector Postscript is preferred to raster Postscript since it may be easily printed, scaled, and encapsulated within other documents with no loss of intrinsic resolution.

3 GENERIC HEP VISUALISATION TOOLS

The most widely used generic physics analysis tools, which require visualisation, are GEANT [3] and PAW [2] (the Physics Analysis Workstation). GEANT is a toolkit for the simulation of the passage of particles through matter. Versions 3 and below are based on Fortran and ZEBRA while GEANT4 [4] is a complete rewrite using an Object-Oriented (OO) paradigm. Since the visualisation aspects of GEANT4 are described by John Allison in these proceedings [5], we shall not consider it further.

PAW is a general purpose tool for the analysis and display of arbitrary data sets. It relies heavily on other HEP packages, in particular: ZEBRA [8] for memory management, I/O and file storage; HBOOK [21] for the creation and manipulation of histograms and ntuples; HPLOT [22] and HIGZ [23] for the data presentation; KUIP [24] for the command line interface and macros; the COMIS [25] Fortran 77 interpreter for user data selection and manipulation functions; SIGMA for array manipulation; and MINUIT [26, 27] for statistical analysis, particularly fitting. PAW, which is based on Fortran 77, was developed without any formal design or quality control procedures. This resulted in frequent occurrences of non-intuitive or incorrect behaviour, especially in the early years. Despite these weaknesses, PAW is the only program inside or outside HEP which comes close to providing physicists with the necessary environment for interactive graphical analysis of physics data. The PAW software is now frozen apart from major bug fixes. The graphical representations provided by PAW match reasonably well to the current expectations of High Energy physicists.

The most commonly used entity is the 1-dimensional histogram in which a dependent variable (such as number of events) is plotted as a function of an independent variable. For example, Fig. 6 shows a comparison of LEP data and Monte Carlo predictions using 1-dimensional histograms. It is conventional to represent the data as polymarkers with error bars, Monte Carlo predictions as a stepped histogram, and analytical predictions from theory as smooth curves. Pie charts and bar charts, in which the independent variable is a set of discrete categories, are little used in HEP. A variant of the bar chart, which we shall call the "error bar chart", is used extensively to summarise sets of measurements of the same quantity (although it is not provided directly by PAW). Fig. 7 shows an example of an "error bar chart" which summarises LEP measurements, at $\sqrt{s} = 172\,\text{GeV}$, of the W boson mass [28]. From the "error bar chart" the relative and absolute accuracies of the various measurements are immediately apparent, as are anomalies with respect to an average value (shown by the shaded vertical bar in Fig. 7)

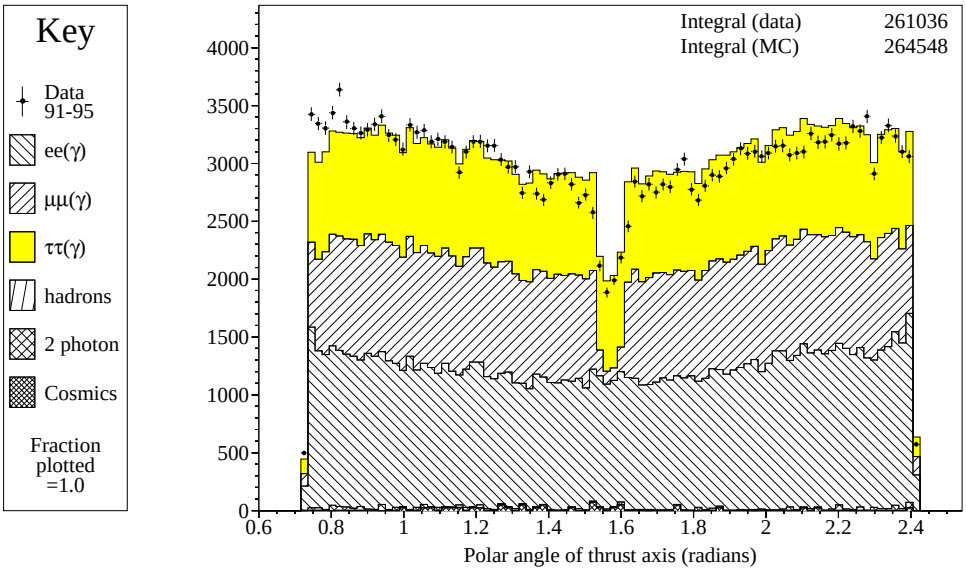

Fig. 6: Example of the use of 1-dimensional histograms to compare HEP data (from LEP) and Monte Carlo predictions.

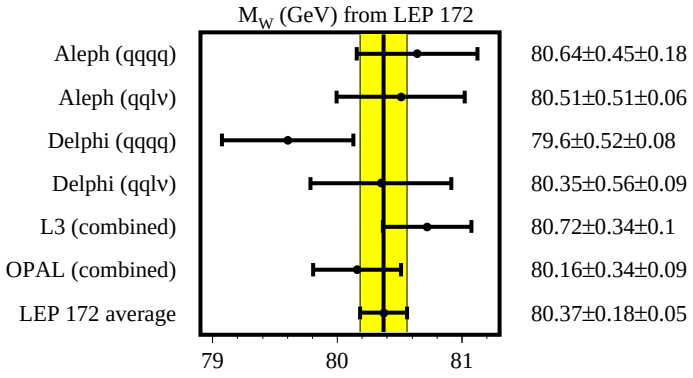

Fig. 7: Example of an "error bar chart", summarising LEP measurements of the W boson mass from running at $\sqrt{s} = 172$ GeV.

or a theoretical prediction.

Two-dimensional histograms, in which one dependent variable is studied as a function of two independent variables, are widely used to study correlations of pairs of variables. Fig. 8 shows an example of various possible representations of the number of events, in a LEP I data sample, having jets with energies E_1 and E_2. The scatter plot (Fig. 8a) has a marker for each event and therefore loses no information until markers start to obscure one another. It is good for low to medium statistics. The box plot (Fig. 8b) is better for medium-high statistics although the scaling of the box is ambiguous (by perimeter or area?) both in definition and in interpretation. The contour plot (Fig. 8c) joins regions of equal value in the independent variable by polylines. It is best for high statistics and reasonably smooth variations without a large dynamic range. The independent variable may also be represented by the height of a box or surface (Figs. 8d, 8e, and 8f) above the plane of the independent variables. Since some parts of the plot may be obscured from view, these representations benefit from an interactive display in which the orientation of the axes may be varied. The independent variable may also be mapped to a colour scale (Fig. 8g). Since there is no unique mapping of colour to the scale of the independent variable, a key is imperative. In certain cases, a combination of a colour plot and a lego or surface plot may be effective (Figs. 8h and 8i).

The representations of one- and two-dimensional data described are those most commonly employed in HEP. Many fields profit from other representations, as discussed by Terry Hewitt [29], which in some cases provide insight into multi-dimensional relationships. While some of these could doubtless be used to the advantage of HEP, time is required for those in the field to develop a natural intuition for them. In general, graphical representations should not to be misleading – they should match the expectations of the audience. For example, the axis scales should be chosen according to the region of interest while avoiding suppressed zeroes. The bin size should be appropriate to the statistics of the histogram. Non-uniform binning should generally be avoided since the distribution of the independent variable is distorted, unless the independent variable is normalised to the bin size. Errors bars frequently have both statistical and systematic contributions which are combined in quadrature. The showing of, for example, the statistical and total errors as pairs of overlapping errors bars in each bin should be avoided since the systematic contribution will appear to be smaller than it is in reality. The use of a 2-dimensional graphic (i.e. one in which both the height and the width scales with the independent variable) for the display of 1-dimensional data such as a histogram should also be avoided, since it over-emphasises the larger entries. Similarly, colour should be used sparingly, if at all, to distinguish and/or emphasise particular aspects of the data.

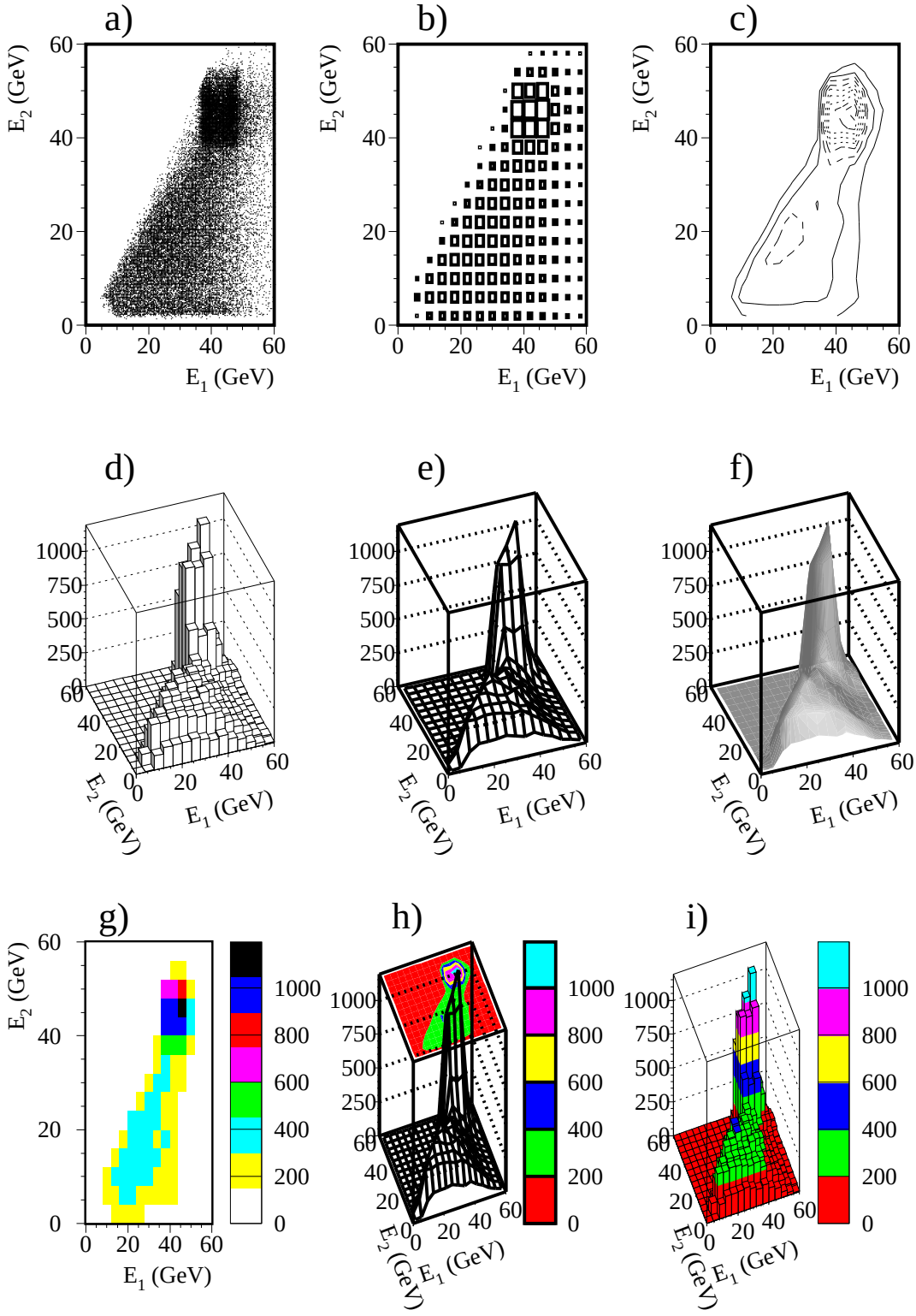

Fig. 8: Representations of the variation of a dependent variable (number of events) on two independent variables (E_1 and E_2, which happen to be correlated): a) scatter plot; b) box plot; c) contour plot; d) lego or Manhattan plot; e) surface plot; f) Gouraud-shaded surface plot; g) colour-coded box plot; h) surface plot with coloured contour plot; i) lego or Manhattan plot with colour-coded contour levels.

4 EVENT DISPLAY SYSTEMS

An "Event Display" program comprises software which is capable of displaying detector and event data and of providing interactive control of the display and processing. Such programs satisfy many different needs. Prior to the running phase of the experiment, they are used for debugging and optimising the detector design and the reconstruction algorithms, such as pattern recognition in the central tracker, shower reconstruction in the calorimeters, and muon fitting in the outer detectors. During the commissioning phase of the detector, they are invaluable for understanding and debugging the detector response and, in this role they must be sufficiently flexible to cope rapidly with unforeseen needs. As more data are collected the display programs are used to study samples of rare events to check for pathological effects. Throughout the lifetime of the experiment, such programs are used to produce publicity pictures for communicating complicated physics concepts to other physicists and the general public.

The event visualisation program should be able to display any combination of detector elements and event objects which are available in the data store. The event data should include raw information such as hits and energy deposits, reconstructed objects such as tracks and jets, and information from the slow control system, such as dead or noisy detector elements. For Monte-Carlo events, both the true and the reconstructed quantities should be available. The engineers designing the detector need to see details of the mechanical construction, the readout cables, cooling system, *etc.* whereas the physicist would prefer to display highly simplified, yet accurate, representations of the detector components in order to place the event data into context. The sub-detector expert may need to see every hit in a calorimeter module, whereas the average physicist would generally find clustered energies and reconstructed jets more useful. The display program must therefore permit sophisticated filtering of information thereby enabling arbitrary sets of elements to be displayed.

The general flow of an event display program is shown in Fig. 4. The steps involved in the visualisation process are described in the following sections.

4.1 Data Preparation

The data preparation stage consists of "data pre-processing", "event reconstruction", and "selection". Pre-processing consists of the conversion of the raw event data, such as ADC counts, using calibration data into basic entities such as hits. In addition, the detector description is read from a database, such as a GEANT structure, and the elements to be visualised are selected and possibly transformed to a more suitable coordinate system. Reconstruction consists of: pattern recognition (e.g. the association of hits to form tracks, clusters of energy, etc.); the creation of basic reconstructed "objects" (e.g. the fitting of hits to form tracks, association of clusters to form jets, etc.); and the creation of higher level reconstructed "objects" (e.g. the association of a track and a cluster to form an electron candidate). Finally, the selection algorithms choose events of particular interest as well as the "objects" within each event which are to be displayed.

4.2 Visualisation Mapping

The next step is to perform the visualisation mapping to create "Abstract Visual Objects" (AVO's). While a few experiments work in two dimensions from the outset, most work predominantly with a 3-dimensional coordinate system corresponding to the cartesian world coordinate system of the detector. Some have dedicated non-cartesian views, such as the energy lego plots as a function of pseudo-rapidity and azimuthal, in hadron collider experiments.

The detector AVO's generally consist of sets of polyhedra, conic sections, etc. which correspond to the more interesting elements from a physicist's point of view, such as the sensitive elements, bulk absorbers, and so on. Some of the event data AVO's are purely geometrical. For example tracks which describe the trajectory of a particle through the detector are readily described by a polyline or spline. Other event data AVO's have additional data associated with them which must be displayed, such as the

energy deposited in a calorimeter, in which case there are many ways of constructing the corresponding AVO's. For example, calorimeter energies may usefully be described by: uniformly sized markers, colour-coded according energy; as volumes representing the detector element hit, perhaps scaled or coloured according to energy; or as histogram bars located at the point of deposition (as shown in Fig. 9). As in the case of the box plot, there is some arbitrariness as to whether the linear length, area, or volume of the AVO should scale with energy. One might even consider other scaling laws such as power law or logarithmic, depending on the dynamic range of energies of interest.

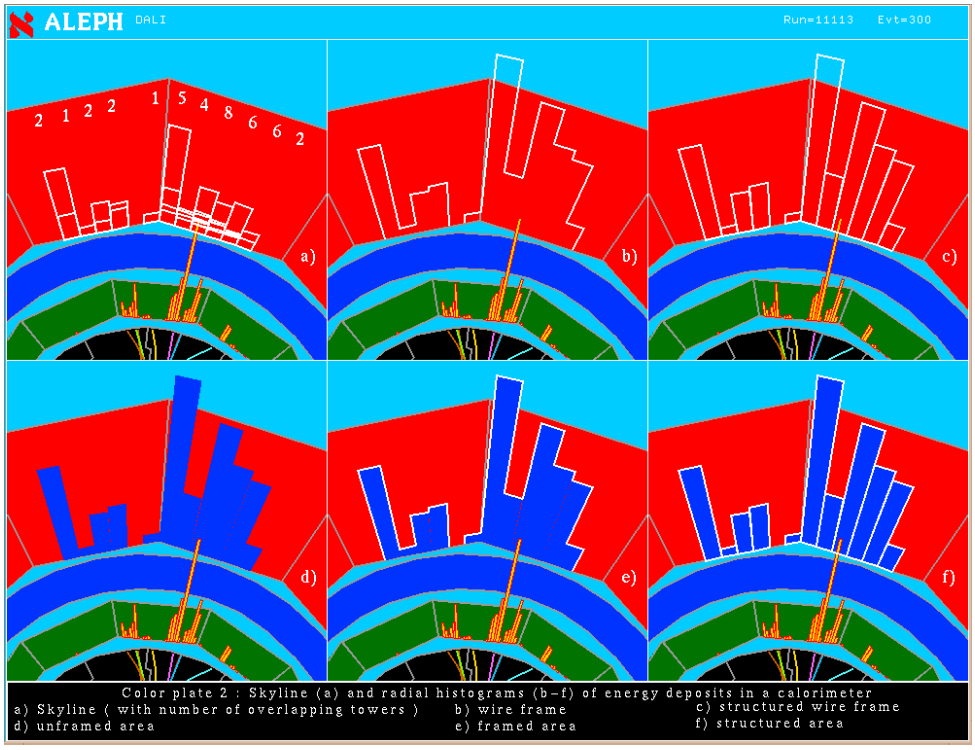

Fig. 9: Different ways of display energy deposits as a histogram (courtesy of H. Drevermann of ALEPH).

4.3 Presentation

The presentation step consists of rendering the AVO's to form a picture, following the general steps described in section 2.4. The single most important view is the 3-D cartesian representation of the detector and event data, with arbitrary magnification, translation, rotation, clipping, and perspective. The view is projected into 2-D although the future implementation of true 3-D, for example using stereo views, should be considered.

Solid volumes, such as the detector elements, are traditionally drawn as wire-frames. This is partly for historical reasons of performance and partly to enable the observer to see through the whole picture which would otherwise be impenetrable. As performance has improved, some event display programs now draw solid volumes and exploit techniques such as clipping planes, cutaways, and transparency to see the otherwise hidden volumes. Although the AVO's are generally 3-D they are ultimately rendered on a 2-D device. Graphics systems invariable project the view from 3-D into 2-D, however, this is not always the optimum choice from a physics standpoint. Consider, for example, viewing energies in a cylindrically symmetric detector with the axis along the beam direction (a common situation). The 3-D representations of the energy clusters may be projected graphically into the xy plane, as shown in Fig. 9a). Alternatively, one could first integrated the energies over z and then display the resulting sums,

as shown in Fig. 9b). Since the choice of representation is not unique, the event display program should be sufficiently flexible to support multiple visual representations of the same quantity, depending on the specific needs of the user.

Visualisation does not consist merely of 3-D cartesian views (projected into 2-D). Physics data are intrinsically multi-dimensional and transformed views of data play a crucial role in our ability to assimilate important features [29–31]. For example, hits on high momentum tracks are separated in x, y, z but are tightly clustered in the ϕ versus θ view, while jet structures are clearly seen in a lego plot of energy as a function of ϕ and pseudo-rapidity. The physicist may imagine many useful non-Euclidean, Lorentzian, *etc.* transformations and a display program should be prepared to support the resulting views. The LHC beam crossing time of 25 ns renders inter- and intra-event times crucial for triggering and reconstruction. This adds a "fourth dimension" which needs to be considered in the context of event visualisation.

Colour is used in HEP event displays for various purposes. In a few cases it is purely cosmetic while in most it is used to distinguish between sub-detector systems and different elements of the event. The continuity of colour usage within a given experiment enables physicists to rapidly interpret the context and key features of events. It is also used to facilitate the correlation of graphical entities between different views of the same event, or even between the display and the GUI controls [32]. Figs 9b)-9f) show how colour (grey-scale for these proceedings) and outlining can be used to improve the clarity of the image.

4.4 Graphical User Interface

A well-designed GUI greatly facilitates the use of event display programs by minimising the learning time, reducing errors, and increasing the efficiency with which complicated command sequences can be executed. Current event display programs in HEP typically use X Windows with the Motif widget set [19] which yields a familiar style. Fig. 10 shows the OPAL event display program, the features of which are fairly typical of current programs in HEP. The programs generally provide basic steering and I/O control, local and global interactivity, and multiple output graphics windows containing cartesian views of the detector and the event as well as abstracted views such as the lego plot of energies in the calorimeter.

4.4.1 Local Interactivity

Fig. 11 shows the motion widget used for manipulations in the "CMSCAN Classic" CMS event display program, which is typical of the local geometrical operations desired. Smooth rotations about three axes, translations, and zooming, may be performed with arbitrary step sizes using either the intuitively labelled buttons or by dragging the mouse across the screen. Commonly used views such as the xy-plane or a zoom to a given sub-detector are also provided. The slice view excludes all objects outside a given clipping plane, parallel to the screen, which has arbitrary thickness and depth location. The possibilities to reset the view and to save and restore the parameters describing the view have proven to be extremely useful.

Typically control of which elements are visible is performed locally at the level of the display list, under the control of the GUI. In some experimental environments, such as CMS, the overhead associated with including all possible AVO's in the display list is extremely large. The strategy is therefore to add AVO's only upon request, such that a request to see a new AVO is a global operation which requires the raw data and methods. Once the AVO is added to the display list its visibility control becomes a local operation. The GUI controls do not distinguish between such global or local operations such that the only difference which is apparent to the user is the response time.

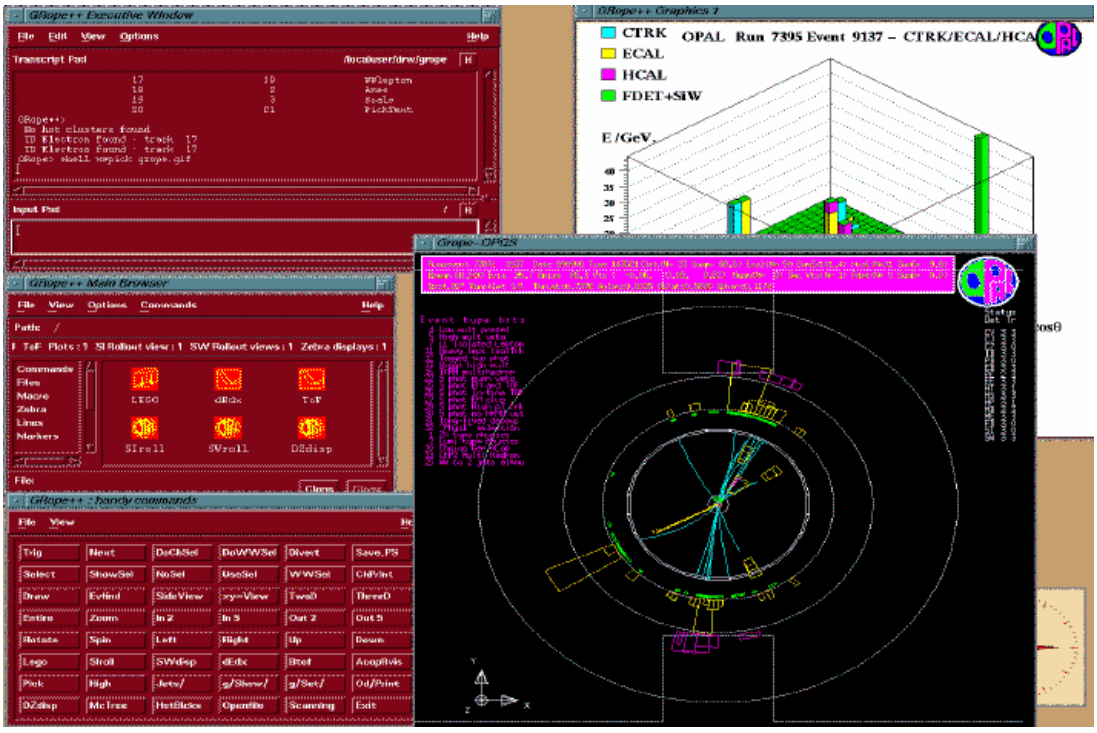

Fig. 10: The graphical display of the OPAL event visualisation program.

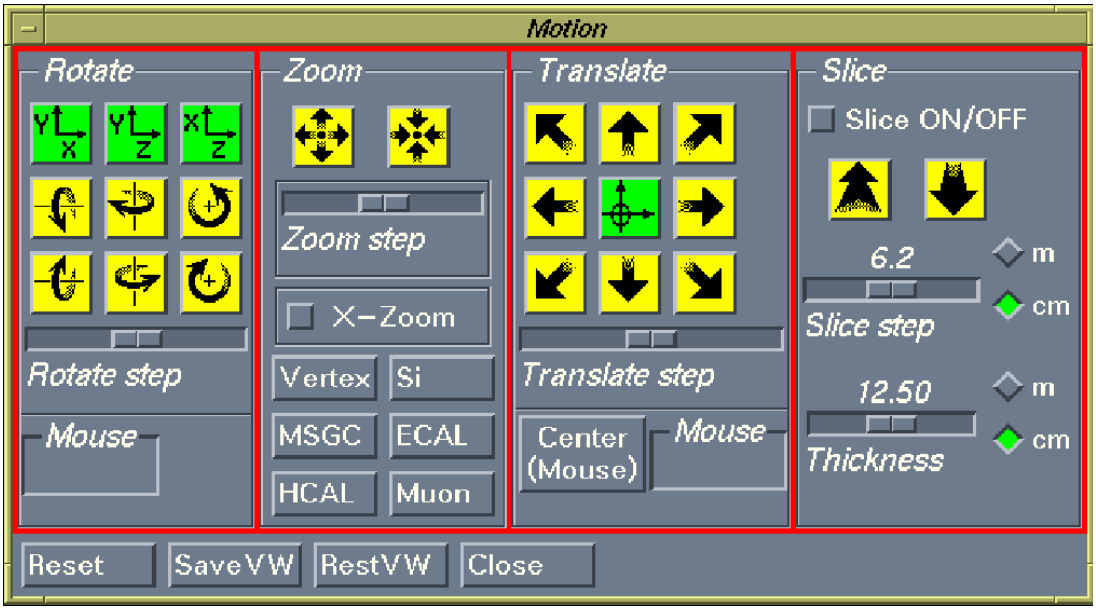

Fig. 11: The CMS motion widget showing local geometrical transformations.

4.4.2 Global Interactivity

The event display program should be more than just a viewing program. Much of its value comes from the ability to interact with the data displayed on the screen with more *global* functionality than merely *local* manipulation of the graphical image [33]. The program should serve as an interface to the data and reconstruction code and support such actions as: application of selection cuts (e.g. energy, p_{T}, event timing, trigger, *etc.*); plotting (e.g. hit residuals with respect to a fitted track, lego plots (as shown in Fig. 10), *etc.*); re-reconstruction (e.g. track fitting, jet finding, *etc.*); kinematic analysis (e.g. invariant masses, Lorentz transformations, *etc.*); statistical analysis (e.g. fitting, hypothesis testing, *etc.*); database interrogation (e.g. trigger conditions, HV status, dead/noisy regions, *etc.*). Where appropriate, the results of any such action should cause an automatic and consistent update of all displayed views. The interfaces between the various software modules should support sophisticated bi-directional interactivity between the graphics and the data and the rapid implementation of new tasks, perhaps even at run-time.

The design and implementation of such global functionality is probably the hardest task in the creation of an effective event display program. Many powerful commercial tools, such as the OpenInventor viewers, concentrate on the local aspects of viewing an image and provide little support for effective interaction with the underlying data and methods from which the view was derived. Neglect of the global aspects of event display in the design phase can result in a program which, although capable of producing beautiful pictures, is of little use as an interactive physics analysis tool.

5 FUTURE DIRECTIONS FOR HEP VISUALISATION

The environment of future HEP experiments is generally much more demanding than that of High Energy Physics experiments today. For example, the number of readout channels for the LHC experiments will increase by typically one or two orders of magnitude to $\sim 10^7$ compared to the LEP experiments. The event data will also be considerably more complicated. Each beam crossing will contain typically 20 minimum bias events, in addition to the event of potential interest, such that each crossing contains many hundreds of charged tracks. The beam crossing time of 25 ns is so short that signals from several crossings are present in the detector and the readout chain at any particular moment. To compound these difficulties, the events of interest are extremely rare. For example, prior to any trigger or offline selection, only one event in $\sim 10^{12}$ is expected to contain an observable Higgs boson decay. In addition to the increasing intrinsic complexity of the visualisation tasks, the organisational aspects of the software are becoming more challenging. Future large HEP collaborations consist of up to several thousand scientists around the world, working on a multitude of hardware platforms.

5.1 Hardware strategy

The hardware requirements of the LHC experiments are on an unprecedented scale. Fig. 12 shows schematically and quantitatively the layout of the proposed CMS computing systems [34]. The data rate will be in excess of 1 PB/year and the total CPU requirement will be approximately 20 TIPS. Such requirements will be met at a reasonable cost by virtue of the decreasing unit costs with time [34], shown in Fig. 13.

The evolution of computer graphics systems is expected to have a similar cost profile. In the early 1980's the UA1 collaboration used the Megatek system for 3-D event display at an approximate cost of $\sim$ 200 kSF. Around 1989, L3 used Apollo DN10000 3-D graphics systems at a cost of $\sim$ 70 kSF. Today, similar performances can be achieved on a multitude of Unix workstations for $\sim$ 10 kSF. A clear trend, which has already started, is to exploit developments in commodity hardware developed for such concerns as the games industry, in particular PC's with on-board graphics acceleration, using for example the OpenGL chip-sets from Intergraph, 3-Dlabs, or 3dPro. It is clearly prudent not to tie applications to a particular hardware architecture or vendor, nor should the developer assume that all users will have a high-end machine. It is, however, desirable to design applications which can take advantage of the

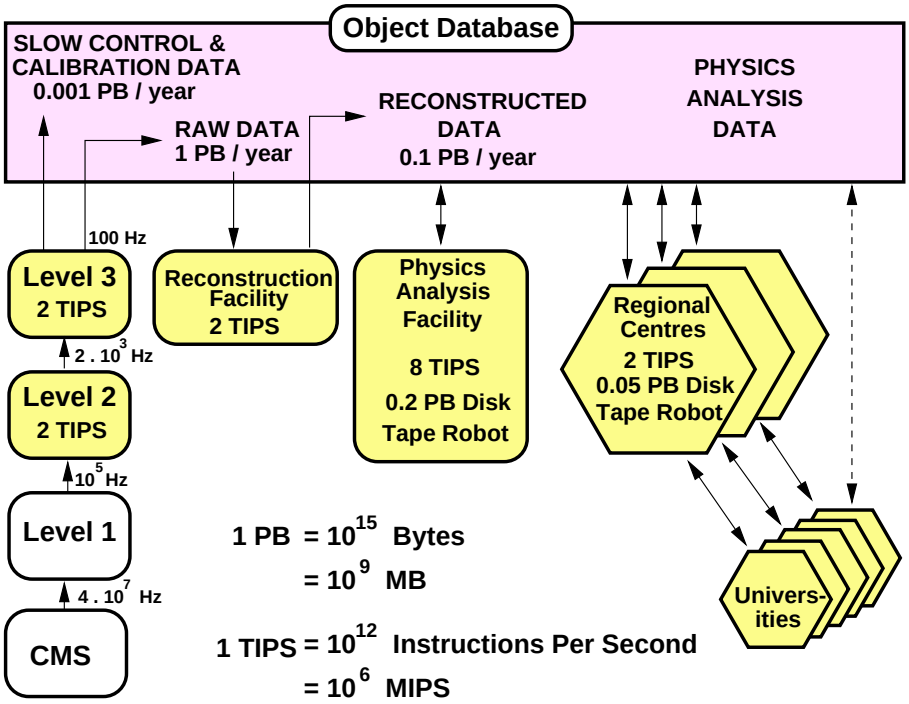

Fig. 12: Schematic layout of the proposed CMS computing systems.

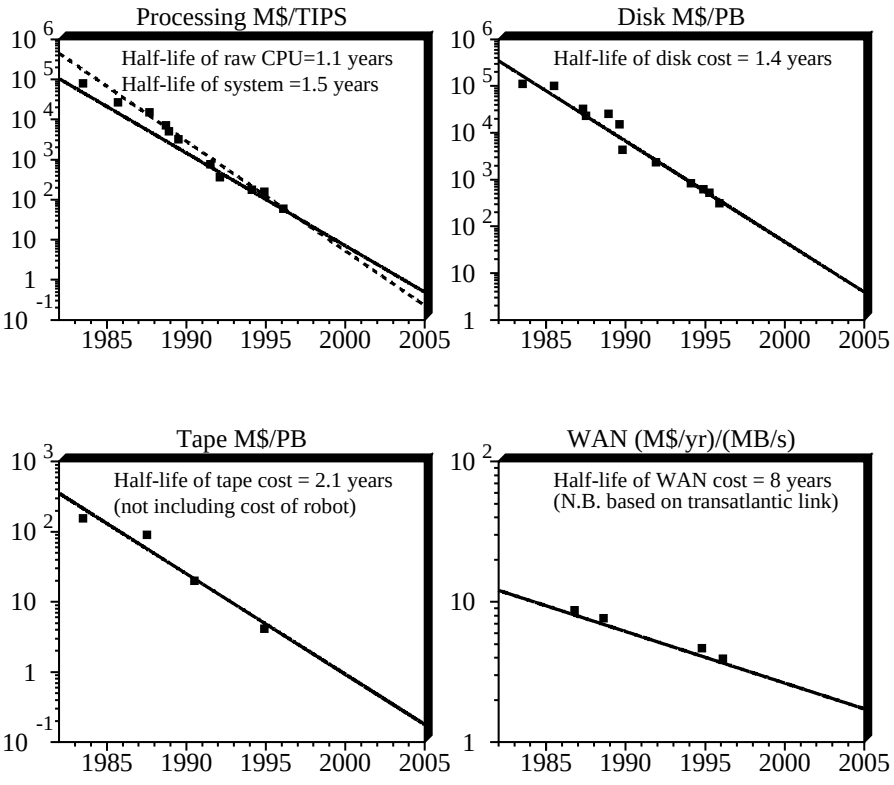

Fig. 13: Evolution of computing hardware costs. The polymarkers denote actual purchases by the L3 collaboration (courtesy of R. P. Mount).

enhanced capabilities of more powerful graphics stations while still performing acceptably on more modest machines.

Despite the ever improving cost:performance ratio, it is unlikely that graphics systems in 2005 will provide acceptable performance without a careful choice of what to display. High-end desktop graphics systems today can render $\sim 10^6$ triangles per second. Given that real-time interactivity requires a refresh rate in excess of about 10 frames/sec, this limits the current model size to less than $\sim 10^5$ triangles. Models of the LHC experiments can contain in excess of 10^7 elements. Assuming that the rendering performance follows a similar trend as CPU, as shown in Fig. 13, then one would predict approximately two orders of magnitude increase in speed by 2005. This would enable one to render a model of less than $\sim 10^7$ triangles which is considerably less than 10^7 elements. Moreover, not everybody will have access to a high end graphics desktop system. Clearly, the answer to this "problem" is not to show all the elements at any one time. This requires the developer and physicist to make choices of what to display at any particular time. In any case, an attempt to display more elements than the number of pixels on the screen is rather futile.

5.2 Software strategy

The software of the LHC experiments will be developed and used over a period of several decades. The design and maintenance of this software poses major problems for the collaborations. The software strategy of most large future experiments at CERN, FNAL, SLAC, and elsewhere is to migrate to an Object Oriented paradigm for the complete offline software by the end of the century [34–36]. The physics generator, simulation, reconstruction, analysis, and calibration software will be written in C++ for the foreseeable future. The use of other languages, such as Java, is not ruled out although all "modern" languages, including C++, suffer somewhat from the lack of stability and well-defined standards.

The use of common-HEP modules and commercial software components is encouraged wherever they fulfil the requirements; development of code by physicists should preferably be restricted to tasks specific to High Energy Physics. Fig. 14 shows schematically the modules required for the offline physics analysis environment, together with the estimated contributions to each module from the experiment, the HEP community, and commercial vendors. Central to the model is a (largely commercial) object data base and mass storage system [37–42] which contains event and slow control data, parameters describing the detector, results from test beam studies, and so on. The software engineering, information system, and data presentation software is also expected to come predominantly from commercial companies. The experiment-specific DAQ, simulation, reconstruction, and analysis software will rely heavily on HEP-wide software such as GEANT4 [4, 5] and the LHC++ [43, 44] libraries.

A detailed strategy for the event visualisation software is difficult to define since commercial applications are developing very rapidly. The current aim is to use commercial software for rendering, development of the graphical user interface [45], and for the underlying graphics library functions. For the latter, the obsolete GKS and PHIGS libraries will certainly be replaced by more modern products [46] such as the OpenGL graphics library, the Open Inventor class library (which in turn uses OpenGL), or newer products based for example on VRML, and Java.

For event visualisation, the display and interactivity pertaining to the detector and events is likely to be based on the GEANT4 and HEPVIS [47, 48] HEP-wide projects. The task of replacing PAW is more difficult, both in terms of the functionality and the prevalence of use. The LHC++ strategy [43, 44] is to develop a generic physics analysis and visualisation toolkit of modules for use within a commercial Modular Visualisation Environment [7], the current choice being Iris Explorer [49]. Wherever possible, commercial components will be used in preference to a "HEP-grown" solution, for example for the rendering and data presentation modules.

Looking even further ahead one might envisage a unified physics analysis environment [50] in which the event visualisation is homogeneously integrated with modules for statistical, numerical, sym-

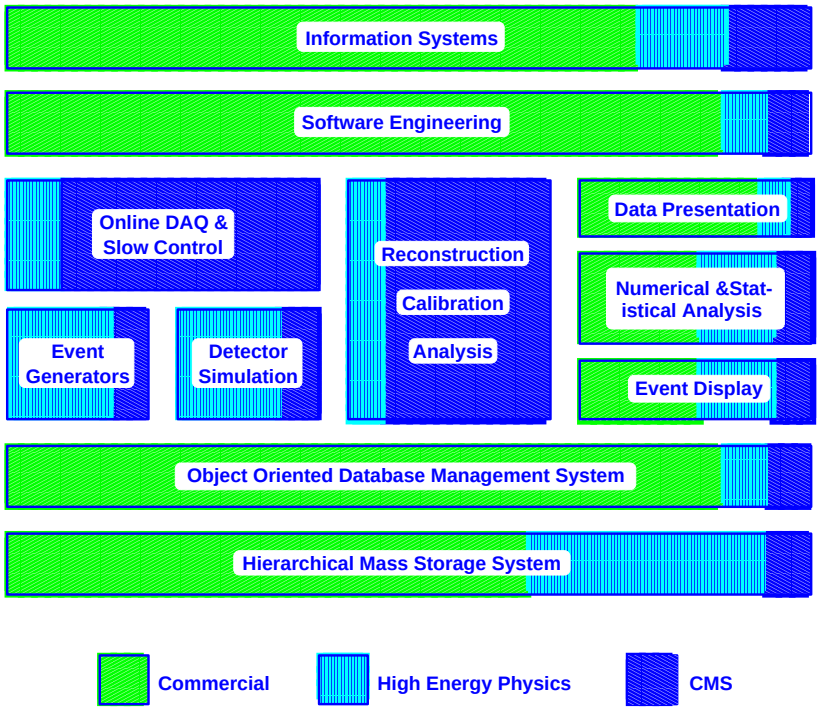

Fig. 14: Software modules for the CMS experiment in an Object Oriented paradigm. The shading indicates approximately the relative quantities of experiment-specific, HEP-wide, and commercial software.

bolic, and user analysis. This could make use of the current MVE model or perhaps a fully distributed paradigm based, for example, on Java [46,51–53]. Given the speed of commercial software developments in recent years, however, the only safe prediction for the future is that it is unpredictable.

6 CONCLUSIONS

Visualisation programs fulfil an invaluable role in the analysis of large multi-dimensional physics data sets, in the design and debugging of the detector, readout, and software, in the evaluation of event reconstruction algorithms, and in the production of understandable images for communicating complex physics concepts to the physics community and the general public.

The detectors and event data of future HEP experiments are considerably more complex than current experiments, the signal events are generally extremely rare and the backgrounds are immense. At the LHC, the short beam-crossing time causes event pile-up which adds a new time dimension to the three spatial dimensions. The visualisation programs need to support arbitrary choices of visibility and imaginative new transformations of the data and the ways in which they are displayed. They need to be flexible with well-designed interfaces to the reconstruction code and data structures. These aims will be more easily achieved using a software-engineered OO paradigm and commodity software components wherever possible. This strategy should enable physicists to concentrate on their field of expertise – namely physics.

ACKNOWLEDGEMENTS

I would like to thank the organisers, lecturers, and participants for such a well-organised, informative, and congenial school. I am grateful to the many colleagues who, sometimes unknowingly, greatly facilitated the preparation of these lectures, especially John Allison, George Alverson, Joe Boudreau, Olivier Couet, Mark Dönszelmann, Hans Drevermann, Terry Hewitt, David McNally, Howard Stone, and John Swain.

REFERENCES

[1] J. Bunn. Collaborative Computing Environments for HEP. In *Invited talk at the CHEP'97 Conference*, Berlin, Germany, April 7-11 1997. `http://www.ifh.de/CHEP97/chep97.html`.

[2] R. Brun et al. PAW Physics Analysis Workstation. CERN/CN Long Write-Up Q121, 1989. (see also: `http://wwwcn.cern.ch/asdoc/Welcome.html`).

[3] R. Brun et al. CERN-DD/78/2, 1978. (see also: `http://wwwcn.cern.ch /asdoc/Welcome.html`).

[4] S. Giani. GEANT4: a world-wide collaboration to build Object Oriented HEP simulation software. In *Invited talk at the CHEP'97 Conference*, Berlin, Germany, April 7-11 1997. `http://www.ifh.de/CHEP97/chep97.html`.

[5] J. Allison. GEANT4 Experience. In C. Vandoni, editor, *Invited lectures at the CERN Computing School*, Pruhonice (Prague), Czech Republic, 17-30 August 1997.

[6] J. D. Foley and A. Van Dam. *Fundamentals of Interactive Computer Graphics*. Addison-Wesley, 1982.

[7] W. T. Hewitt. Systems and Architectures for Visualisations. In C. Vandoni, editor, *Invited lectures at the CERN Computing School*, Pruhonice (Prague), Czech Republic, 17-30 August 1997.

[8] R. Brun and J. Zoll. ZEBRA user guide. CERN–CN Long Writeup Q100.

[9] V. Blobel. The BOS System. DESY R1-88-01, 1988.

[10] H.J. Klein and J. Zoll. PATCHY Reference Manual. Available from CN division, CERN, 1988.

[11] CERN and CodeME S.A.R.L. CMZ – A Source Code Management System. Available from CN division, CERN, 1990.

[12] See, for example: `http://www.sdsu.edu/doc/texi/cvs_toc.html`.

[13] F. R. A. Hopgood et al. *Introduction to the Graphical Kernel System – GKS*. Academic Press, 1983.

[14] D. R. Myers. GKS/GKS-3D Primer. CERN/DD/US/110, 1990.

[15] T. Gaskins. *PHIGS Programming Manual*. O'Reilly & Assocs. see also: PHIGS Standard: ISO/IEC 9592-1,2,3:1989(E); 9593:1989(E).

[16] See, for example: `http://www.sgi.com/Products/Dev_environ_ds.html`
See also: T. Davis, Jackie, Neider, M. Woo, *OpenGL Programming Guide: The Official Guide to Learning OpenGL*, Addison-Wesley, Massachusetts, (1993).

[17] See, for example: `http://www.sgi.com/Technology/Inventor.html`.

[18] A. Nye and T. O'Reilly. *X Toolkit Intrinsics Programming Manual, OSF/Motif 1.2 Edition*. O'Reilly & Assocs., 1992.

[19] D. Heller and P. Ferguson. *Motif Programming Manual*. O'Reilly & Assocs., 1994.

[20] Adobe Systems Incorporated. *Postscript Language Reference Manual, 2nd ed*. Addison Wesley, Reading, MA, USA, 1990.

[21] R. Brun and D. Lienart. HBOOK User Guide. CERN/CN Long Write-Up Y250, 1987. (see also: `http://wwwcn.cern.ch/asdoc/Welcome.html`).

[22] R. Brun and N. Cremel Somon. HPLOT User Guide. CERN/CN Long Write-Up Y251, 1988. (see also: `http://wwwcn.cern.ch/asdoc/Welcome.html`).

[23] Application Software Group. HIGZ and HPLOT User's Guide. CERN Program Library Writeups Q120 and Y251, 1992. (see also: `http://wwwcn.cern.ch/asdoc/Welcome.html`).

[24] R. Brun and P. Zanarini. KUIP – Kit for an User Interface. CERN Long Write-up I202, 1988. (see also: `http://wwwcn.cern.ch/asdoc/Welcome.html`).

[25] V. Berezhnoi et al. COMIS – Compilation and Interpretation System. CERN–CN Long writeup L120, 1988. (see also: `http://wwwcn.cern.ch/asdoc/Welcome.html`).

[26] Application Software Group. MINUIT Reference Manual. CERN/CN Long Writeup D506, 1992. (see also: `http://wwwcn.cern.ch/asdoc/Welcome.html`).

[27] F. James. Interpretation of the Errors on Parameters as Given by Minuit. Supplement to CERN–CN Long Writeup D506, 1978.

[28] L. Taylor. Properties of the W Boson. In *XVIIth International Conference on Physics in Collision*, Bristol, UK, 25–27 June 1997. To be published.

[29] W. T. Hewitt. Visualisation of Multidimensional and Multivariate Data. In C. Vandoni, editor, *Invited lectures at the CERN Computing School*, Pruhonice (Prague), Czech Republic, 17-30 August 1997.

[30] H. Drevermann, D. Kuhn, and B. S. Nilsson. Is there a Future for Event Display? In *Proceedings of the 1992* CERN *School of Computing*, L'Aquila, Italy, September 1992.

[31] H. Drevermann et al. Imaginative Transformations for High Multiplicity Events. In L. Taylor and C. Vandoni, editors, *Proceedings of the HEPVIS'96 Workshop*, CERN, Geneva, Switzerland, September 2-4 1996. CERN 97-01 (Yellow Report).

[32] P. R. Avery C. D. Jones and D. D. Roscigno. The Cleo3D Event Display. In L. Taylor and C. Vandoni, editors, *Proceedings of the HEPVIS'96 Workshop*, CERN, Geneva, Switzerland, September 2-4 1996. CERN 97-01 (Yellow Report).

[33] H. Stone. Event Visualisation – a Physicist's perspective. In L. Taylor and C. Vandoni, editors, *Proceedings of the HEPVIS'96 Workshop*, CERN, Geneva, Switzerland, September 2-4 1996. CERN 97-01 (Yellow Report).

[34] The CMS Collaboration. Computing Technical Proposal. *CERN/LHCC*, 96-45, 1996. ISBN 92-9083-096-4.

[35] The ATLAS Collaboration. Computing Technical Proposal. *CERN/LHCC*, 96-43, 1996.

[36] M. Mazzucato. The LHC Computing Model. In C. Vandoni, editor, *Invited lectures at the CERN Computing School*, Pruhonice (Prague), Czech Republic, 17-30 August 1997.

[37] D. Duellmann. HEP Data analysis based on an ODBMS store. In L. Taylor and C. Vandoni, editors, *Proceedings of the HEPVIS'96 Workshop*, CERN, Geneva, Switzerland, September 2-4 1996. CERN 97-01 (Yellow Report).

[38] RD45. Object databases and Their Impact on Storage-Related Aspects of HEP Computing. *CERN/LHCC*, 97-7, 1997.

[39] RD45. Using an Object Database and Mass Storage System for Physics Analysis. *CERN/LHCC*, 97-9, 1997.

[40] J. Shiers (representing RD45). Objectivity/DB Performance and Scalability. In *Invited talk at the CHEP'97 Conference*, Berlin, Germany, April 7-11 1997.

[41] RD45. Project Execution Plan. *CERN/LCB*, 97-10, 1997.

[42] P. Binko. OO Databases. In C. Vandoni, editor, *Invited lectures at the CERN Computing School*, Pruhonice (Prague), Czech Republic, 17-30 August 1997.

[43] LHC++ - Libraries for HEP Computing: `http://wwwcn.cern.ch/asd/lhc++/index.`

[44] J. Shiers. LHC++ - "CERNLIB" for LHC? In *Invited talk at the CHEP'97 Conference*, Berlin, Germany, April 7-11 1997. `http://www.ifh.de/CHEP97/chep97.html`.

[45] G. Cosmo. Graphical User Interfaces. In L. Taylor and C. Vandoni, editors, *Proceedings of the HEPVIS'96 Workshop*, CERN, Geneva, Switzerland, September 2-4 1996. CERN 97-01 (Yellow Report).

[46] See, for example: `http://hpl3sn02.cern.ch/taylorl/hepvis/products.html`.

[47] The HEPVIS Project. See: `http://cactus.phyast.pitt.edu/~joe/hepvis/hepvis.html`
See also: Proceedings of the HEPVIS'96 Workshop, Eds: L. Taylor and C. Vandoni CERN, Geneva, Switzerland, September 2-4 1996.

[48] G. Alverson et al. The HEPVIS Class Library for Event Visualization. In *Invited talk at the CHEP'97 Conference*, Berlin, Germany, April 7-11 1997. `http://www.ifh.de/CHEP97/chep97.html`.

[49] The Numerical Algorithms Group (NAG), "IRIS Explorer Center", `http://www.nag.co.uk:70/Welcome_IEC.html`.

[50] J. Swain and L. Taylor. Data Analysis – a Physicist's perspective. In L. Taylor and C. Vandoni, editors, *Proceedings of the HEPVIS'96 Workshop*, CERN, Geneva, Switzerland, September 2-4 1996. CERN 97-01 (Yellow Report).

[51] A. Johnson. Distributed Data Analysis using JAVA. In *Invited talk at the CHEP'97 Conference*, Berlin, Germany, April 7-11 1997. `http://www.ifh.de/CHEP97/chep97.html`.

[52] M. Dönszelmann. WIRED - World-Wide Web Interactive Remote Event Display. In *Invited talk at the CHEP'97 Conference*, Berlin, Germany, April 7-11 1997. `http://www.ifh.de/CHEP97/chep97.html`.

[53] M. Dönszelmann and B. Rousseau. Information Systems for Physics Experiments. In C. Vandoni, editor, *Invited lectures at the CERN Computing School*, Pruhonice (Prague), Czech Republic, 17-30 August 1997.