CMSCAN Classic

George Alverson
Lucas Taylor

Northeastern University, Boston

1 CMSCAN Classic

The current version of the CMS event display program is labeled 'Classic' because it relies on classic technologies. The detector and its response are generated using GEANT3 [2] (written in FORTRAN), all widgets are based on those present in the Motif widget set, and the graphics package is PHIGS [6], an industry standard for 3D. These components were picked as an interim choice until suitable stable equivalents are available in either C++ or Java, and in fact no additional development is anticipated for this version of CMSCAN (although work is continuing on new versions [5]). The point of discussing such a standard system lies in the fact that its graphics system replaced an earlier package and that this replacement offers us a picture of the procedure we will face in moving to OO-based graphics and beyond.

2 CMSCAN Features

As a mature program [1, 3, 4], CMSCAN has a variety of features to ensure ease of operation and to provide the full functionality required of an event display:

1. Full 3D control of detector orientation: widgets allow for rotation around any of the x, y, or z-axes. There is also a mini-mousepad for mapping 2D mouse motions into 3D.

2. Preset views for often used magnifications and orientations: rather than just a zoom widget, we have individual widgets for particularly useful magnifications.

3. Z-Slice: since the occupancy is so high for LHC detectors, this widget limits the data limited to that in a certain z range. The scales available are appropriate for the segmentation actually present in the detector.

4. Independent or ganged x and y magnification: this is particularly useful when examining the deviation of a track from its constituent hits.

5. Text annotation widget: allows the user to put in labels using point-and-click.

6. Vector Postscript output (either direct or by CGM metafile to PS utility).

7. Visibility control of sub-detector and sub-event elements: again, given the complexity of the detectors and events at LHC, this is an absolute necessity. We have broken these down into logically coherent units and control their visibility with a custom-designed widget.
8. Simultaneous multiple views: these allow the user to have up to four different views of the detector/event simultaneously visible. Each view can be manipulated independently of the others, providing control of both the orientation and which components are visible.

9. Picking: only local interactivity is presently provided. (Here we use local to mean querying only information contained within the graphics object and not referring the query back to the detector/event routines. See H. Stone [8].)

You may note what we don’t have: a command line interface. This was a deliberate design decision in the light of our experience with other event display programs.

3 Design Decisions

As with seemingly all modern software projects, our initial design decisions were driven by manpower economics: our intention was to use as much standard software as we possibly could and to provide only the glue and whatever customization was necessary for ease of use. GEANT3 had already been chosen as the detector modeling package, so we were left with a choice of the graphics and interface packages. We decided to pick a standard GUI, Motif, as it was available on all our systems and has good support. For the graphics we chose PHIGS, again because it had good support at CERN and seemed to be a leading industry standard which would not be quickly abandoned, and also because it provided a good palette of 3D primitives.

The first problem was how to remove the graphics normally associated with GEANT3 (HIGZ [7]) and replace it with PHIGS. The HIGZ interface was 2D and PHIGS is 3D, so merely writing an interface layer to translate HIGZ calls to PHIGS calls would have been fruitless. Instead we had to dig back up into the graphics package of GEANT3 itself (see Figure 1). Luckily the 3D information we needed to create 3D objects was kept until the final stage so this was not as difficult as it might have been.

Unfortunately, we found that the overall performance was unacceptable. This required a step back to analyze the problem, and resulted in a new streamlined version of a few of the GEANT3 internal routines involved in detector drawing. We also learned how to delay the overhead involved in drawing a particular section of the detector until the first request for it was made (via the visibility control), rather than all during initialization. This made the delay more palatable to the average user. (We should stress once again that for an environment less complicated than the LHC, as at LEP, these measures are not required, although useful.)

4 The Frame Structure

When we discovered that the visibility of various portions of the display would have to be frequently changed, we decided that this demanded a careful solution that would be robust and efficient. In trying to achieve a
design that would also be generic enough to be easily modified, we hit upon modeling the scene as a separate directed acyclic graph (to use a jargon term denoting a tree structure without loops) which paralleled the equivalent structure in PHIGS. Our copy, however, would only keep nodes which were significant to the user and thus be quick to traverse. To distinguish it from the PHIGS structure, we called ours a frame.

The frame was built within the program with a series of simple structure-creating commands. Each node in the frame had a pointer to a routine (typically in C but occasionally in FORTRAN) which was responsible for creating the parallel PHIGS object or objects. This very simple and modular design was mirrored in the control widget. It also became very easy to make a copy of the frame, which gave rise to our multiple view capability (see Figure 2).

5 Conclusion

From our experiences with CMSCAN Classic we drew a number of conclusions which we hope will enable us to do a better and quicker job as we make our move from Classic software technologies to more object oriented ones. Let’s start by listing what we wanted when we began:

- Fast: should respond to the user without hesitation
- Uniform: should operate in the same manner on all platforms
- Clean: minimal changes in code between platforms
- Minimal: use existing software

In reality, what we ended up with had little resemblance to the ideal. Instead we had to resort to a variety of hacks and tricks and hard work in order to get adequate performance. We

- implemented special 2D projections: 3D objects were collapsed to 2D (this also reduces the size out the size of the Postscript output tremendously);
- implemented (by hand) a poor person’s Level of Detail: “small” objects were represented with less detail;
- (optionally) represented solid objects as line sets;
• customized our interface to match our detector and physics;
• used selective visibility to keep the number of objects to a minimum.

We also used (as an option) a vendor specific version of PHIGS on HP machines. We found that a Sun Sparc Ultra 3D Creator with G5G PHIGS performed roughly on a par with an HP 712 with HP PHIGS. This was not a pleasant experience: vendor-specific PHIGS extensions are not uniform, as we discovered when we tried to port to Digital PHIGS.

We thus see that although we strive for generic and modular programs, we often must break these rules for the sake of performance. The lesson we take on with us is to keep the interior of our packages as clean as the interfaces, because it will most likely need to be modified in some unforeseen way (assuming that we’ve written a good enough package that someone would want to use it).

References


