Event display for the International Linear Collider Summer student report

Stewart Martin-Haugh, supervisor Steve Aplin
Physics Computing/IT

September 10, 2009

1 Introduction

The International Linear Collider (ILC) is a proposed high-energy physics experiment, and will be the second machine (after the LHC) to effectively probe the TeV energy scale. It is an e^+e^- collider, with $\sqrt{s} = 500\text{GeV}$ maximum centre of mass energy. Very broadly summarised, the ILC will be able to make precision measurements of new physics discovered at the LHC, with the cleaner experimental profile offered by an e^+e^- collider [1]. Many decisions about the experiment have yet to be made: the most important of which are where and when it will be built, and which detector prototype it will adopt. There are three proposed detector prototypes: ILD, SiD and Fourth. In order to evaluate the physics case for the machine, and distinguish between the different detectors, an unprecedented level of initial analysis using full Monte Carlo data has already been performed. This undertaking means that much of the ILC software is mature at a far earlier stage than for previous experiments. Visualisation is important already at this stage, in order to check the validity of reconstruction, simulation, and to explore the different strengths and weaknesses of the detector concepts. The processing chain is shown in figure 1.

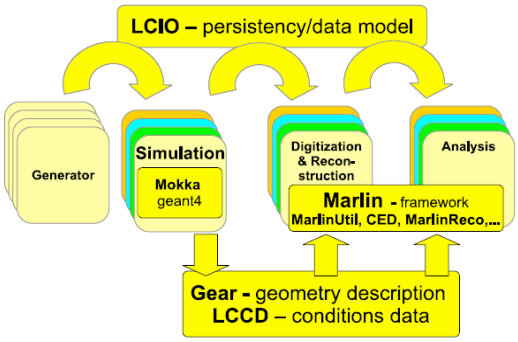

Figure 1: Overview of the ILC software framework and processing chain.

2 The Generic Event Viewer

One of the early tasks of the project was to evaluate whether a newly developed event display might be suitable for the ILC. The event display in question, GenericEventViewer, is (as of the time of writing) being developed by Andreas Moll [3] and colleagues at the Max Planck Institute for Physics in Munich. It is written in C++ and uses the QT 4 framework for its graphical user interface. It currently offers some attractive features, in particular its modularity and modern choice of tools. Since the ILC is a long-term project, it is important, at this stage, to choose standard computer software likely to be available and supported over the next few years. In the version used for evaluation, it was possible to show tracks and geometric primitives, but this was not yet extended to allow standard ILC event and geometry files to be read. Once this functionality is in place, and an initial release has been made, it would be interesting to re-evaluate the software.

3 The C Event Display

The next focus of the project was the C Event Display (CED), so-called because of its choice of programming language. It operates using a simple client-server communication model, which will be discussed in detail later. It is lightweight and relatively simple , but its main shortcoming is that it offers no graphical user interface, and cannot be customised at runtime. One of the goals of this project was to investigate the viability of a graphical user interface for CED.

3.1 Fisheye Transforms

A HEP detector centred at the interaction point of an accelerator is usually cylindrically symmetric, with small (∼cm) inner detectors ranging to larger outer detectors. This disparity of scale is a problem for the human observer, because it makes difficult to view the hierarchy of interactions at all levels. The ILC detector concepts use the particle flow algorithm [4], in which a single particle is followed through the tracking detectors and calorimeters. The relative importance of a detector element is not proportional to its size. In fact, different detector elements are more significant for different particles. This is only possible if we adopt a non-Euclidean coordinate system for the event display, exaggerating the size of the inner detector relative to the outer detectors. A popular example of this is the "fish-eye" transform, given [5] as follows:

$$\rho = \sqrt{x^2 + y^2} \tag{1}$$

$$\rho_F = \frac{\rho}{1 + \alpha\rho} \tag{2}$$

$$z_F = \frac{z}{1 + \alpha|z|} \tag{3}$$

$$\phi_F = \phi \tag{4}$$

It is easy to verify that the transform is non-singular at the origin (unlike the choice $z_F = \log(z_F)$, and tends to 1 as the ρ and z co-ordinates tend to infinity. The transform thus corresponds to a mapping of the space $\{0, \infty\}$ onto the space $\{0, 1\}$. Using the absolute value of z and keeping the angle the same allows the cylindrical symmetry

to be maintained. After the transform, the co-ordinate system must be changed back to Cartesian. Two helper functions were written to perform this transform in CED: firstly

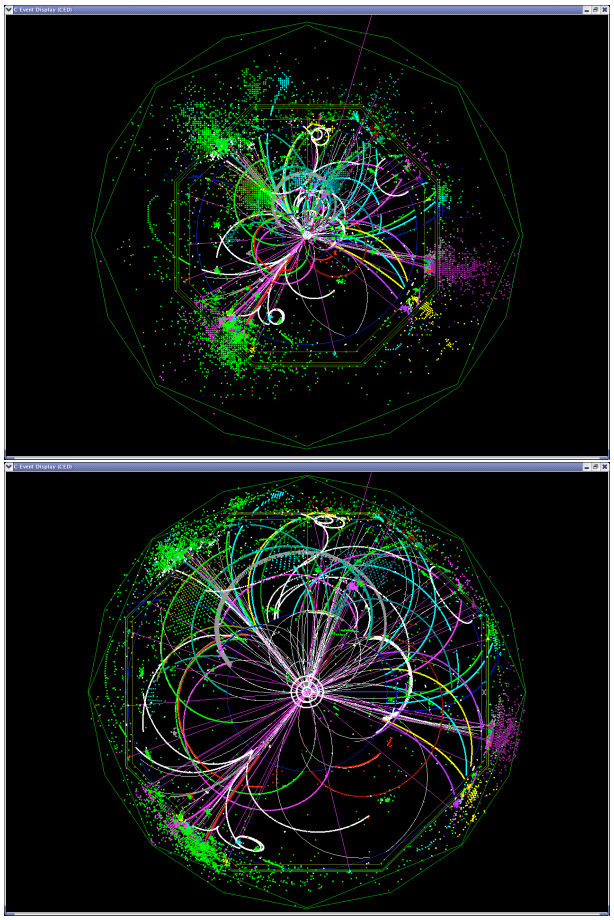

Figure 2: Front view of the ILD for an $e^+e^- \Rightarrow t\bar{t}$ event with (top) and without (bottom) fisheye.

fisheye_transform(), which implements it for an arbitrary x, y, z relative to the beam axis, and a second, single_fisheye_transform(), which implements it for a single co-ordinate only. In addition to this, the ced_draw methods were altered to include the fisheye transform. Altering each function individually to this end means that this is not a general solution. A suggestion for a more elegant implementation may be found in [6]. The idea is that one works directly with the Vertex Buffer in OpenGL, leaving all the ced_draw methods intact, and offering performance improvements, particularly with respect to hardware acceleration. Additionally, this means that new drawing methods could be designed without consideration of the fisheye transform. This modularises the code as well as making it less prone to errors.

3.2 Server-client communication

As it stood before the project, CED (the server) would accept hits, tracks, and detector geometry through the Marlin client (specifically the MarlinCED class). CED was unable to send information back to the client, which is a clear barrier to a graphical user interface,

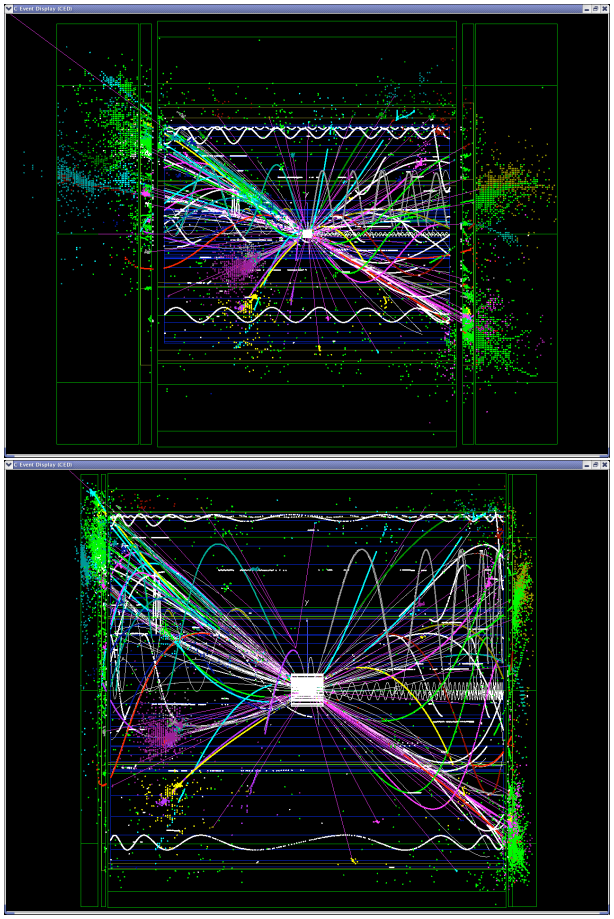

Figure 3: Side view of the ILD for the same event.

for which constant communication in both directions is required. The existing client-server interface uses POSIX sockets [7], so it was decided to adopt this for the server-client communication. The API functions used are given below:

Sockets API functions	
connect()	Open a socket
write()	Send information to a socket
read()	Receive information from a socket
close()	Close a socket

More information is available in[7]. The sample application for this is picking, i.e. selecting a track, hit or geometry (via point and click) and then seeing information about it appear on screen. This is possible since every object displayed is a representation of an LCIO object, and has a corresponding unique LCIO ID number. If this is passed to CED with the drawing commands, then CED can return this information to the client. Code was written to display information about Monte Carlo particles when Monte Carlo tracks, or hits which could be traced back to a simulated source, were displayed. A remaining problem was that once the picking loop was entered, it was not possible to cycle through the rest of the events in Marlin: control was lost meaning that Marlin would have to be shut down.

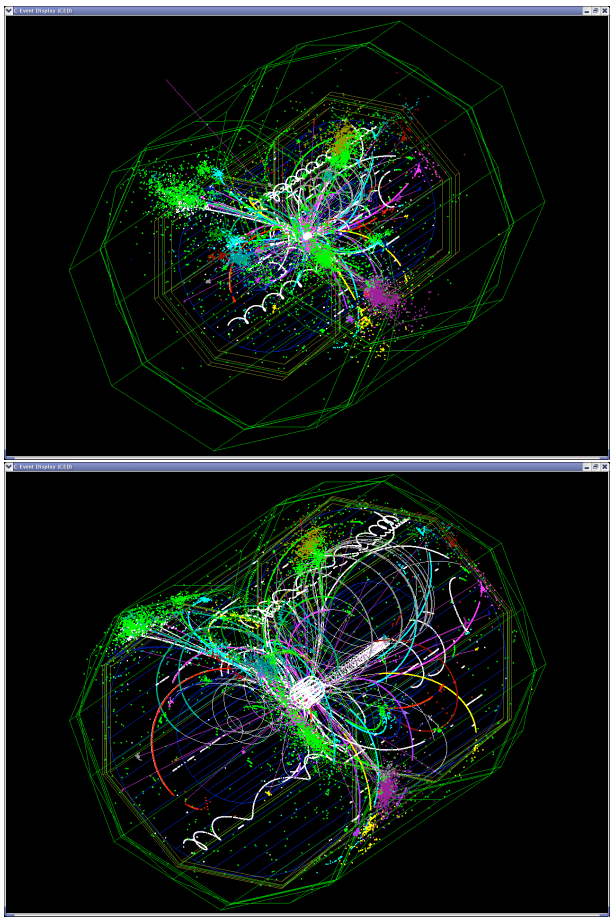

Figure 4: Perspective view of the ILD for the same event. The display may be rotated and translated as usual. Off-axis helices can be seen to be distorted, but this is a feature of the transform itself.

3.3 Helper functions for Marlin

To complement the client-server communication code, helper functions were written for MarlinCED, adding some basic analysis functionality for Monte Carlo Particle (MCParticle) objects. These are intrinsically useful, as well as offering proof of concept for server-client communication.

New helper functions for MarlinCED	
printMCParticle()	Display relevant information about a particle
printMCFamily()	Display information about a particle, its ancestors and offspring
printAndDrawMCFamily()	Same as above, but draw the family too

printMCParticle() provides output in the following format:

```
[    id    ]PDG     |   px        ,   py        ,   pz        |   energy   |gen|[simstat]
[    17739]    -211| -7.54e-01, -8.28e-01,  1.70e+00|  2.04e+00|1   |[   vts]
```

Note that more information (including vertex, endpoint, mass, charge, and no. of daughters) is given in the full version. printMCFamily() is recursively defined: it calls itself for

each daughter but with the number of "branches" reduced by one, and similarly for each mother particle. Since the numbers of branches are both unsigned integers, the function is guaranteed to exit. printAndDrawMCFamily() is very similar, but also draws the particle hierarchy.

3.4 Bug fixes and general improvements

In addition to the work mentioned, some general improvements and fixes to the Marlin and CED source code were made. In particular, the drawing code in CED and the drawHelix() method in MarlinCED were improved. Two problems with the drawHelix() method were found:

- Some particles were not displaying in the correct positions.
- Particles with low momentum were being treated the same as those with high momentum.

The first of these problems was found to be due to an issue with the line drawing function call: it was being called with the argument (z,y,z) instead of (x,y,z). When working with the QWERTY keyboard layout, where 'x' and 'z' are adjacent, this is a very easy error to make, and one that will not be picked up by the software. The second problem was due to the fact that very high momentum charged particles do not curve noticeably in magnetic fields, so may be well approximated for visualisation as straight lines. This was implemented already, but a mistake with the logic meant that the low momentum tracks were also given straight lines. Since very low momentum particles curl very tightly in magnetic fields, the decision was made to approximate them as points. A few trivial changes to Marlin codes were made, standardising the definition of π across several source files.

3.5 Incorporation into Marlin and CED code database

All of the work discussed has bene incorporated into the SVN repository [8]. Because of the problems mentioned above, the picking code was commented out so as not to create problems with the released code.

4 Conclusions and outlook

The work presented here can be seen as a first step towards a more interactive, configurable version of CED. As mentioned above, a future GUI implementation would require server-client communication using sockets, as implemented here for the picking algorithm. Similarly, some extensions to the viewing options in CED will be needed, and one of these, the fisheye view, is now available.

5 Acknowledgments

I would like to thank Steve Aplin, Frank Gaede and Jan Engels for extensive discussions and useful help. Each of them took on some supervision tasks, and made me feel very at

home in the group. I would also like to thank Joachim Meyer and the rest of the DESY Summer Student Programme organisers for a wonderful opportunity.

References

[1] ILC Reference Design Report, Volume I: Executive Summary

[2] ILD Letter of Intent

[3] `http://www.mpp.mpg.de/~molland/GenericEventViewer/`

[4] Innovations in ILC detector design using a particle flow algorithm approach
Stephen R Magill, New J. Phys, 2007

[5] Event Display: Can We See What We Want to See?
H. Drevermann, D. Kuhn, B.S. Nilsson, CERN School of Computing, 1995

[6] OpenGL discussion forum
`http://www.opengl.org/discussion_boards/ubbthreads.php?ubb=showflat&Number=262910`

[7] Open Group Base Specifications Issue 6
IEEE Std 1003.1, 2004 Edition
`http://www.opengroup.org/onlinepubs/000095399/basedefs/sys/socket.h.html`

[8] Marlin SVN Repository
https://svnsrv.desy.de/viewvc/marlin/Marlin/