Development of a visualisation processor in the core software framework for International Linear Collider (ILC) detector

Szymon Daraszewicz[1,*] and project supervisor: Dr. Frank Gaede[2]
[1] *The University of Edinburgh*[†]
[2] *IT Group, Deutsches Elektronen Synchrotron (DESY), Hamburg, Germany*

September 17, 2008

The project primarily aims to develop a Marlin[1] processor (*DTSViewer*) for viewing Monte Carlo generated high energy physics event files, which have no hit information stored; the so-called DST files. The introduction outlines the International Linear Collider (ILC) software chain setting the background for the project. The second section briefly remarks on the usefulness of the DST file format and identifies the need for *DTSViewer* development. Later, a flowchart for *DTSViewer* processor is presented and modifications to the core framework tools such as *CED* and *MarlinCED* are succinctly summarised. Methods for drawing of particle tracks, clusters, jest hypotheses and particle momenta representations are described in great detail. The report concludes with a few suggestions for further improvements as well as reviews the potential virtues of the new viewer.

1. INTRODUCTION - THE NAME OF THE GAME

The software framework for International Linear Collider (ILC) is at the stage of research and development. A typical chain of processing in any high energy physics framework consists of the following steps: generation, simulation, digitisation, reconstruction and analysis; this is schematically shown in Fig. 1. In this report we will only be dealing with visualisation at the reconstruction stage. Throughout, we will be using Marlin, which is an application used for further processing of the data at digitisation, reconstruction and analysis level(1). In the following, I would like to describe a new processor, which uses the data obtained from Marlin to display the reconstructed particles from the so-called DST files.

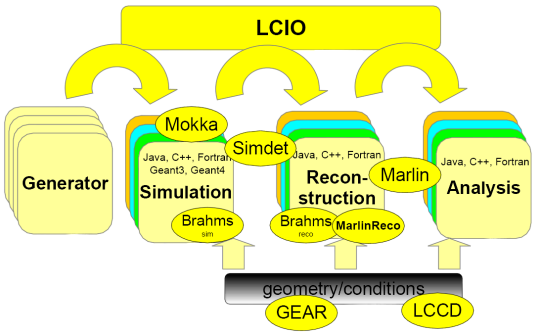


FIG. 1: Software framework tools for high energy physics data - an overview.

2. MOTIVATION AND AIMS

DST is a file format defined at Zeuthen (DESY) for storage of particle collision (i.e. event) information. It stores the following particle collections (i.e. data types): *Tracks*, *Clusters*, *ReconstructedParticles*, *DurhamJets*, *MCParticleSkim* (reconstructed particles & parents, decays in flight & conversions) and finally *LCFIVertex* flavour tag in *ParticleID* objects (i.e. hypothetical particle ID).

Previous Marlin processors such as *CEDViewer* dealt mainly with full-reconstruction files, abbreviated usually as REC. Average size of such files per event is 1800kB. However, if the hit information is removed the files become of a typical size of 23kB/evt. This is the key reason behind the removal of this collection; smaller files mean quicker processing time. In particular, Marlin jobs performed on DST files are two orders of magnitude in time quicker than jobs done on REC files, see Tab. I. Therefore, the next natural step would be the development of a processor, which would deal with DST files in an efficient and a user friendly way. The key requirements of such a processor would be to display *ReconstructedParticle* tracks (lines and helices), particle clusters, Durham jet hypotheses and optionally to visualise the momenta of the particles. The viewer developed in the course of

type	kB/evt	$\nu_{I/O}$ [Hz]
SIM	950	10
REC	1800	3
DST	23	250

TABLE I: Average file size and Marlin job processing time for different file types.

*Electronic address: S.Daraszewicz@physics.org
†DESY Summer School 2008
[1] **M**odular **A**nalysis **R**econstruction for **Lin**ear Collider - modular C^{++} application framework for the analysis of Linear Collider I/O (LCIO) data

this project - *DSTViewer* - meets all of these requirements; the algorithms, which were implemented in it are outlined in section 3.

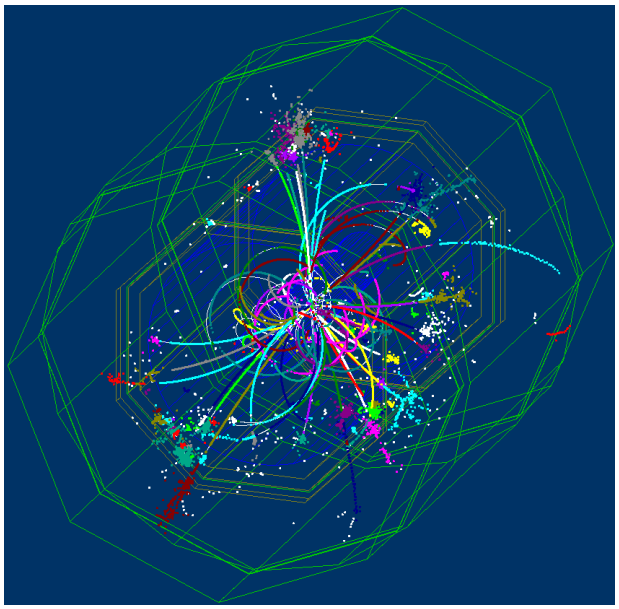


FIG. 2: Rear-view of a fully reconstructed file REC. Surplus of information is apparent.

3. DSTVIEWER - A PROCESSOR FOR DST FILES

Briefly remarking on the overall structure of the code; flowchart view of the *DSTViewer* processor is presented in Fig. 3.

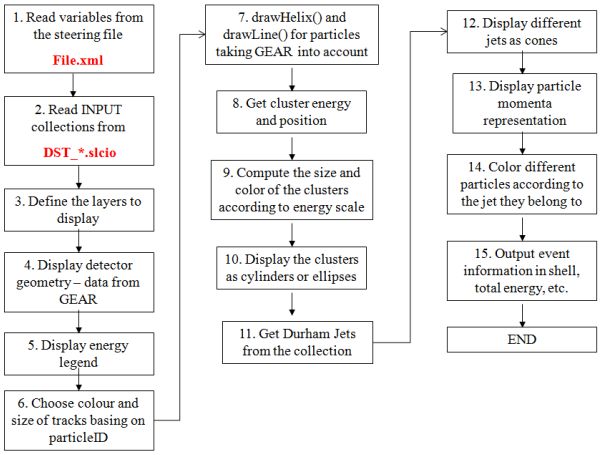


FIG. 3: Key functions flowchart for *DSTViewer*.

The outline of core code of the viewer is as follows.

```
virtual void init()
virtual void process RunHeader (LCRunHeader* hdr)
virtual void processEvent (LCEvent *evt)
\\ code of the DSTViewer goes here...
virtual void check (LCEvent *evt)
virtual void end ()
```

This is in consistency with other processors developed for Marlin.

3.1. Particle tracks drawing

The following methods were implemented for particle track drawing: `drawHelix()` and `drawLine()`, which take care of the charged and uncharged particle drawing respectively[1]. Both methods take on the following variables from GEAR:[2] `float bField` (magnetic field inside the detector in T), the geometry of the detector in the form of `float rmin, rmax, zmax`, which define the tracking chamber geometry (`rmin` may take into account the vertex tracker if set to nonzero value) and the beginning and the end of the TCP[3]. This information allows the particle tracks to be drawn in such a way that they stop at the face of the calorimeter nicely.

`drawHelix()` method currently uses a cheap adaptive-step algorithm, so to draw least possible number of line segments for a reasonably approximated helix. An improvement to use a $versine(\theta)$ function might further reduce the number of line segments drawn per helix at a given 'coarseness'. Limiting the number of line segments drawn per helix is of utmost importance, as a large number of line segments displayed simultaneously may result in '`WARNING:CED: can't send event (...)`' error. Methods `int color = returnTrackColor(type)` and `int size = returnTrackSize(type)` assign color and size of tracks from the hypothetical particle ID obtained by `ReconstructedParticle part->getType()`.

3.2. Clusters - energy representation

Clusters carry information about the energy and the average position of the constituent calorimeter hits of particles. Hence a correct and intuitive graphical representation of these properties is of cardinal importance.

In *DSTViewer* the clusters are represented as cylinders with their direction indicated by a needle inside these; cluster centre is visualised by a star. In the actual code, this is taken care by the `ced_geocylinder_r()`, `ced_hit` and `ced_line` functions implemented in *CED*.

---

[1] `drawHelix()` still can take uncharged particles as arguments and perform as `drawLine()` due to compatibility problems, which will be resolved in the future.
[2] **G**eometry **API** for **R**econstruction
[3] Time Projection Chamber

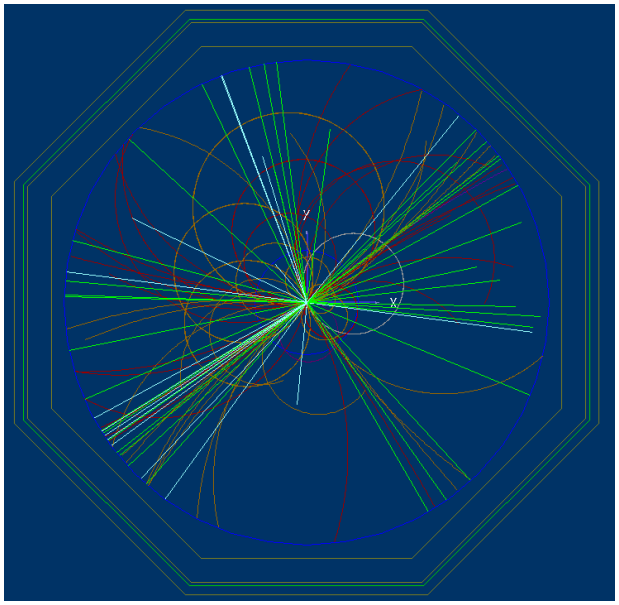
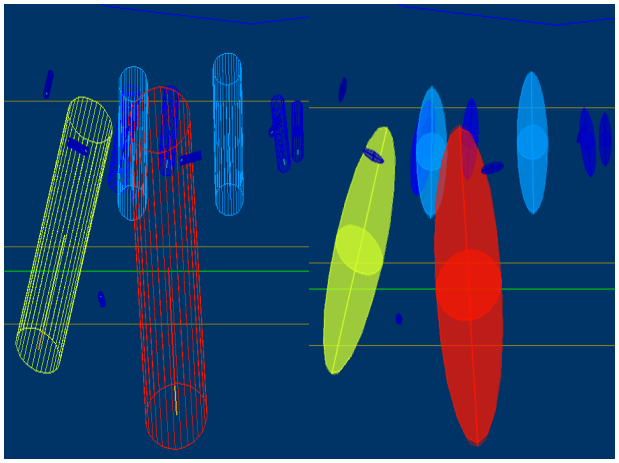


FIG. 4: Particle tracks (both helices and lines) drawn in *DSTViewer*. Front view of the detector.



(a) Cylinder (b) 3 Ellipses

FIG. 5: Different cluster representations compared.

Alternatively one can choose (this is coded on different *CED* layers[4]) the clusters to be represented by 3 orthogonal ellipses. This approach has an advantage that the height and base of the ellipses may represent some additional variables of our choice; see Fig. 5.

It was decided that the cluster size should represent the E_{tot} of the cluster and scale as $\sim log(E_{tot})$. It is the most suitable choice as there is a high spread in the energies of the clusters and hence a linear representation would be ineffective. The following function `DSTViewer::returnClusterSize()` computes the size of the cluster based on its energy, `float eneCluster`. Minimum and maximum energy cut-offs (`float cutoff_min`, `float cutoff_max`) are the other input variables in this function and set the energy scale. Furthermore, energy is coded not only in the size of the clusters, but also in the colour of these. A *DSTViewer* method `returnRGBClusterColor()` takes the cluster energy and the cut-offs as arguments as well as `int color_steps`, which defines the number of colours we would like to use (this is usually set to 256 or 128) and `char scale`, which allows to switch between a linear and a logarithmic energy color-map. Finally, `int colorMap` defines the color-mapping one would like to use. Possibilities include: Jet, Cold and Gray color-maps; see Fig. 7.

Two-dimensional energy colour-map legend `showLegendSpectrum()` with a logarithmic or a linear scale has also been implemented in *DSTViewer*, see Fig. 7

Interestingly, usage of cluster visualisation on REC files may very nicely indicate areas where the clustering algorithm has failed, see Fig. 6.

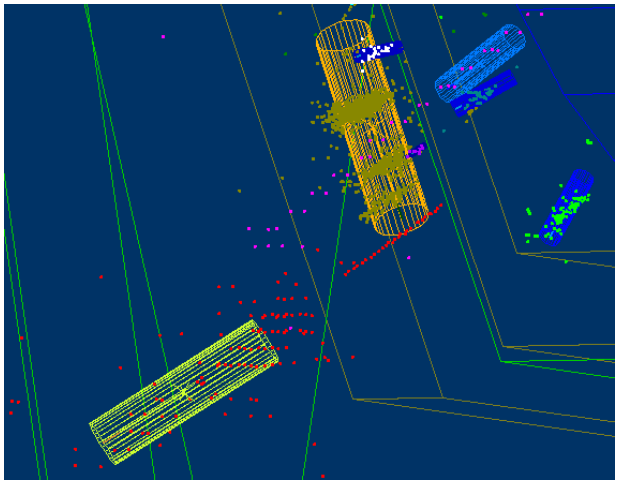
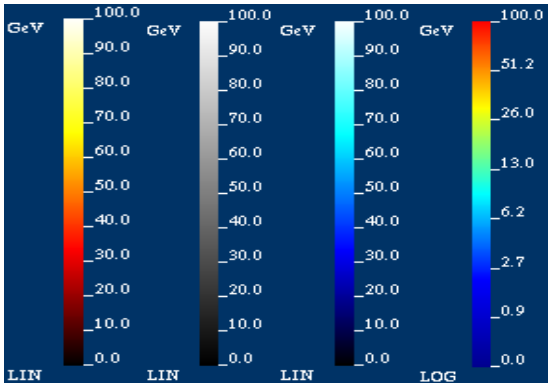


FIG. 6: With the new representation of clusters rare errors in the clustering algorithm are now clearly visible. Note the biggest cluster in the middle; it clearly consists of 3 distinct showers.

3.3. Durham jets hypotheses

Different Durham jets hypotheses were visualised in the code using semi-transparent cones with their apexes at the origin. Transparency was achieved using openGL libraries, namely the following functions: `glEnable(GL_BLEND)` and `glBlendFunc(GL_SRC_ALPHA,`

[4] *CED* layers can be toggled on or off at run-time



(a) Hot (b) Gray (c) Cold (d) Jet

FIG. 7: Legends for different colour-maps. Notice that the Jet colour-map uses a different (log) scale.

`GL_ONE_MINUS_SRC_ALPHA`). Different hypotheses are put on different CED layers and can be - due to the transparency - switched on and viewed simultaneously.

Furthermore, jets in their visualisation encode the following information; the length of the jet cones scales linearly with energy, $L_{cone} \sim E^{jet}_{tot}$. The base of the jet cone represents the total transverse momentum relative to the principal jet axis, $B_{cone} \sim a + \sum_{particles} \vec{pt}$, where a is an arbitrary constant to off-set the size of the base.

Note that the choice of colours for jets was arbitrary, but can be used for representation of another variable in the future (for instance the goodness of the jet algorithm fit).

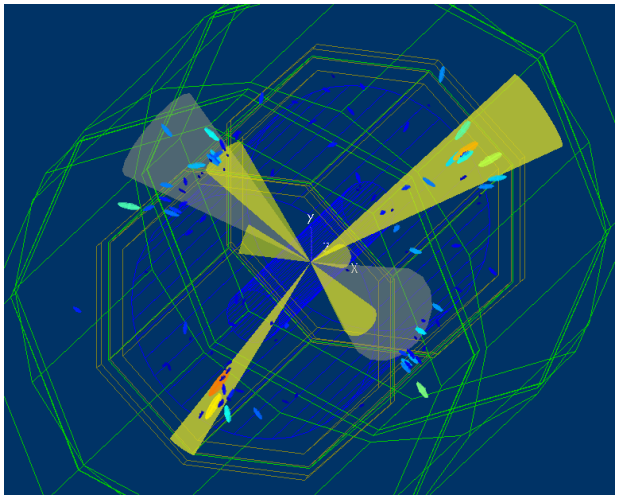


FIG. 8: 6-jet and 4-jet hypotheses visualisation, based on Durham Jet reconstruction algorithm. Note useful information encoded in the colours of the clusters.

3.4. Momenta visualisation

Another useful tool, particularly for jet analysis, which was implemented in the code was particle momenta visualisation. The 3-momenta of the particles are represented as scaled straight lines pointing from the ip[5]. In addition, these are colour coded, according to the particle 3-momentum magnitude.

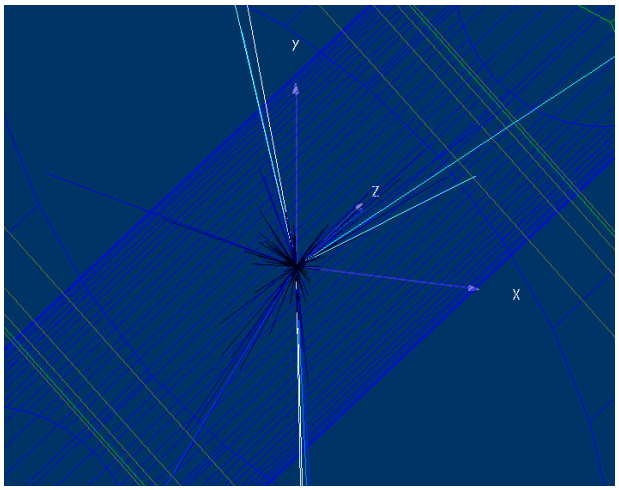


FIG. 9: Particle momenta visualisation; note a large number of particles with insignificantly small momenta. Momentum magnitude of a particle is mapped with a Cold colour-map.

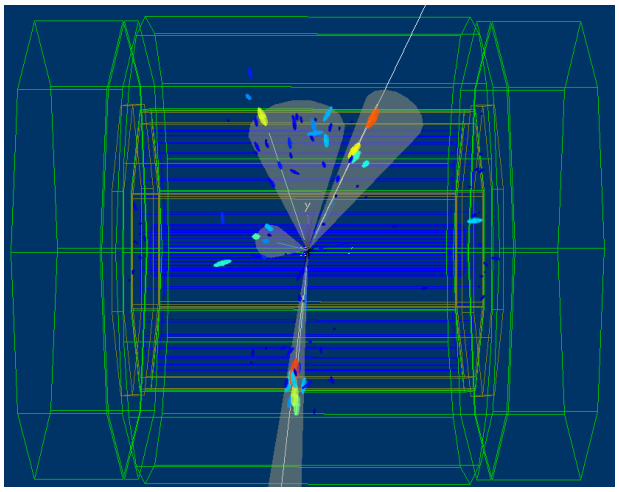


FIG. 10: *DSTViewer* has the potential to become a very powerful tool for the analysis of DST files. The particle momenta drawing in conjecture with jets visualisation provide an excellent and intuitive tool for different jet hypothesis checking. The image presents, clearly, a 4-jet event.

[5] interaction point

4. DOCUMENTATION

Both paper and electronic documentation of the software is of great importance for an end-user, as well as other developers, who would like to re-use the code. Fairly complete documentation was automatically generated from the header files using *doxygen*. In case of any doubts regarding the code functionality please contact the author.

5. SUMMARY AND FUTURE DEVELOPMENTS

In summary *DSTViewer* might become a powerful tool for visualisation and analysis of the DST files for the future ILC related data. The software framework is at a stage of continuous development and will be subsequently improved based on high energy physics community feedback.

There are many areas of potential development in order to improve the code functionality, reusability and package user-friendliness. As said, the current version of *DSTViewer*, which this report refers to, is not the final one. Potential changes may be expected, for example, in the area of particle tracks drawing. Once the medium of the detector and other variables are known `drawHelix()` method could be improved by a Bethe-Bloch correction. Furthermore, on the user-friendliness side, particle picking (once a particle is selected from the *DSTViewer* its properties would be displayed in the shell) remains a feasible option. Finally, a text-based GUI would greatly enhance the end-user experience with the software.

Acknowledgments

First of all, I should like to thank DESY Summer School 2008 organisers, mainly Prof. Dr. J. Meyer, who made it possible for me to carry-out this research at DESY. Undoubtedly, this opportunity maintained my enthusiasm to explore particle physics and perhaps it may open me the door for further academic opportunities.

I am also grateful to my supervisor Dr. F. Gaede for his patience, while making crucial comments regarding the code. I truly appreciate the time and effort he put into the project and inspiration he gave me for further study. I shall also thank Dr S. Aplin and J. Engels for the valuable discussion time and technical support.

Finally, I am enormously grateful to my friends here at DESY, especially '*the container IT group*' for putting up with me for all this time.

References

[1] F. Gaede and J. Engels, *A software framework for ILC detector R&D*, DESY, 2007

[3] F. Gaede *ILD software, towards the LOI and beyond*, ILD Meeting Cambridge, September 11-13, 2008

[3] F. Gaede *Computing in High Energy Physics, an introductory overview*, Summer Student Lecture, August 20, 2008