A Vertex Reconstruction Toolkit and Interface to Generic Objects (VERTIGO)

Winfried Mitaroff and
Wolfgang Waltenberger
Institute of High Energy Physics
Austrian Academy of Sciences
Nikolsdorfer Gasse 18
A-1050 Vienna, Austria, Europe

A proposal is made for the design and implementation of a detector-independent Vertex Reconstruction Toolkit (VERTIGO). It aims at re-using existing state-of-the-art algorithms for geometric vertex finding and fitting by both linear and robust estimation methods; kinematic constraints will also be included for the benefit of complex multi-vertex topologies. The design is based on modern object-oriented techniques. A core (RAVE) is surrounded by a shell of interfaces and a set of analysis & debugging tools. The implementation follows an open source approach and is easily adaptable to future standards.

Motivation and goals

Offline data reduction chain

- The early stages – local pattern recognition, track search and track fitting – are highly detector-dependent;
- whereas the next stage – vertex reconstruction (finding and fitting) – is almost fully detector-independent.
- Vertx fitting with kinematic constraints may be subject to the requirements of a subsequent physics analysis.

Why looking for a toolkit ?

- Geometric vertex finding and fitting must not compromise the high spatial resolution of modern vertex detectors.
- This goal can be achieved by new, sophisticated methods beyond the traditional least squares or Kalman filter estimators, using robust, non-linear, mostly adaptive algorithms – like the deterministic annealing filter (DAF).
- It is not desirable for each new detector to re-code vertex reconstruction software from scratch – provided there exists an adequate, reliable and easy-to-use TOOLKIT.

A good point to start from

- In an initiative called "The RAVE Manifesto", one author *(WW)* proposes taking out vertex reconstruction from the CMS general reconstruction software ORCA, thus providing the basic stock for the core of such a toolkit.
- However, the core must be complemented by flexible interfaces and a modular set of analysis & debugging tools.

General design considerations

The RAVE core

- Collection of the best algorithms available for vertex reconstruction – finding, fitting, kinematics.
- Starting with the packages developed by CMS, but open for entries by other parties (e.g. ZVTOP).
- Code to be based on C++ and HEP-wide (not CERN-specific) OO standards. Well-designed abstraction for interfacing with the outside world will be the key of success.

Shell of interfaces

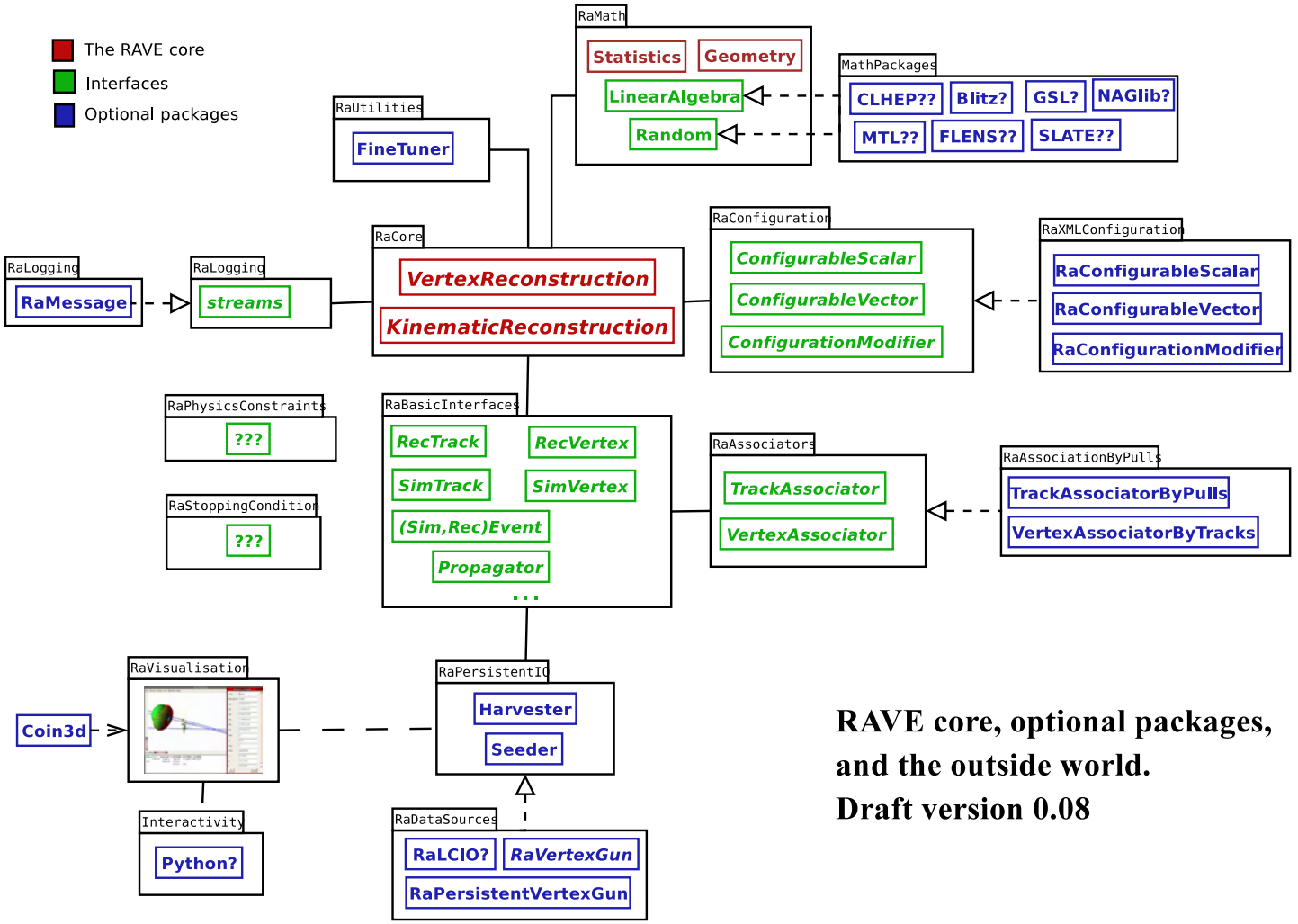
- Access from/to the outside world will exclusively proceed via a shell of interfaces surrounding the core.
- These interfaces make use of adaptors in order to keep a high level of abstraction.

Analysis & debugging tools

- Optional packages, containing those parts of code which might be helpful without being strictly necessary.
- Prototypes of a few packages have already been written:
  - Framework for a stand-alone realisation of VERTIGO,
  - Persistency storage solution ("data harvester" concept),
  - DataSources ("vertex gun", interfaces to LCIO etc.),
  - Visualisation tool (based on COIN3D and PYTHON),

  but much more work is still to be done.
- Extensive use of open standards will minimize the burden of development for this part of the toolkit.

VERTIGO overall design – draft



RAVE core, optional packages, and the outside world. Draft version 0.08

The RAVE core (1)

RAVE = "Reconstruction Algorithms for Vertices"

The following list of candidate core algorithms is by no means final. At present, it is dominated by algorithms either implemented in or investigated for the CMS offline reconstruction (ORCA).

General packages

- `VertexPrimitives` : *Pascal Vanlaer (IIHE Brussels)*
- `VertexTools` : *Pascal Vanlaer*
- `ClusteringTools` : *Wolfgang Waltenberger*

Vertex fitting

- `LinearizationPointFinder` : *Wolfgang Waltenberger (HEPHY Vienna)*
- `LinearVertexFit` : *Pascal Vanlaer*
- `KalmanVertexFit` : *Thomas Speer, Kirill Prokofiev (Univ. Zurich)*
- `RobustVertexFit` : *Wolfgang Waltenberger*
- `MultiVertexFit` : *Wolfgang Waltenberger*
- `GaussianSumVertexFit` : *Thomas Speer*
- `KinematicFitter` : *Kirill Prokofiev*

The RAVE core (2)

Vertex finding

- `PrincipalVertexReco` : *Pascal Vanlaer*
- `AgglomerativeVertexReco` : *Wolfgang Waltenberger*
- `VQuantVertexReco` : *Wolfgang Waltenberger*
- `DetAnnealVertexReco` : *Nicolas Estre, Eric Chabanat (IN2P3 Lyon)*
- `SuperFinder` : *Wolfgang Waltenberger*
- `SuperParamagneticClusterer` : *Wolfgang Waltenberger*
- `KinematicFinder` : *N.N.* ?

Non-CMS packages

- `ZVTOP` : *David Jackson (RAL Oxfordshire)*

New entries of first-class algorithms are highly welcome.

Documentation

Documentation (based on Doxygen) of the algorithms, including information about their scope of application, will be provided. The proper choice of algorithms is also supported by the SKIN concept.

RAVE algorithms – two examples

Vertex fitting by the DAF

The **Deterministinc Annealing Filter** (DAF) is an **adaptive fitter** with an **annealing schedule**. It is implemented in `RobustVertexFit`. Robistification w.r.t. "outlier tracks" is achieved by iteratively downweighting the contribution to the least-squares ansatz of each track's reduced residual r_i by an extra weight w_i, given by a Fermi function with cutoff parameter r_{cut} and "temparature" T (iteration index k omitted):

$$w_i(r_i, T) = \frac{e^{-r_i^2/2T}}{e^{-r_i^2/2T} + e^{-r_{cut}^2/2T}} = \frac{1}{1 + e^{(r_i^2 - r_{cut}^2)/2T}}$$

Iterations start with $w_{i,1} = 1$ (least-squares). For $k > 1$, the $w_{i,k} = w_i(r_{i,k-1}, T_k)$, with $T_k \leq T_{k-1}$ defined by the annealing schedule. For $T \to 0$, the Fermi function approximates the Heaviside function, and the adaptive "soft" track association turns into a deterministic "hard" one ($w_i = 1$ or 0).

Vertex finding by Apex Points

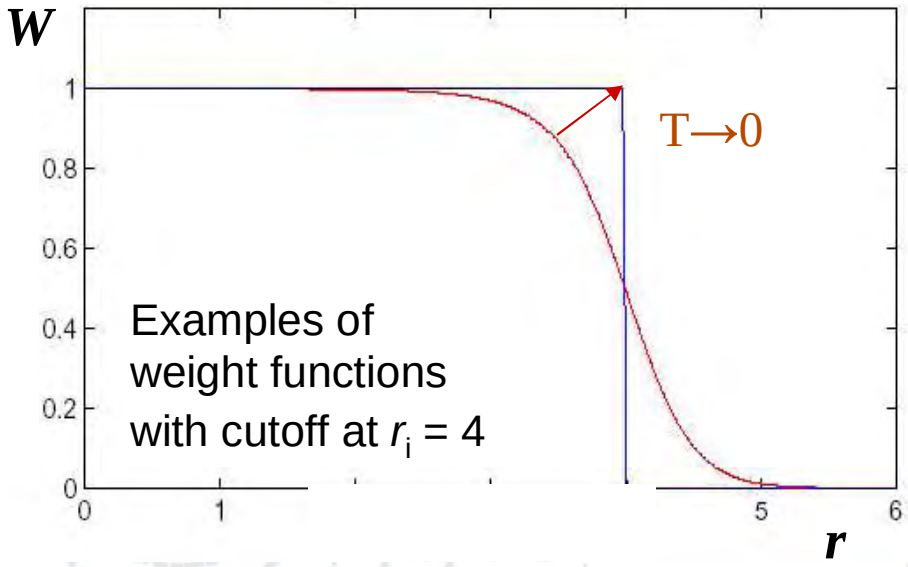
An "apex point" fully represents a track w.r.t. the vertex finding problem at hand. `ClusteringTools` implements an **apex point finder** based on various algorithms (HSM, MTV, MAM), all of which yield a 3d-space point with error matrix on the track. These apex points are then used for vertex finding, e.g. by a **hierarchic, agglomerative clustering** method (implemented in `AgglomerativeVertexReco`).

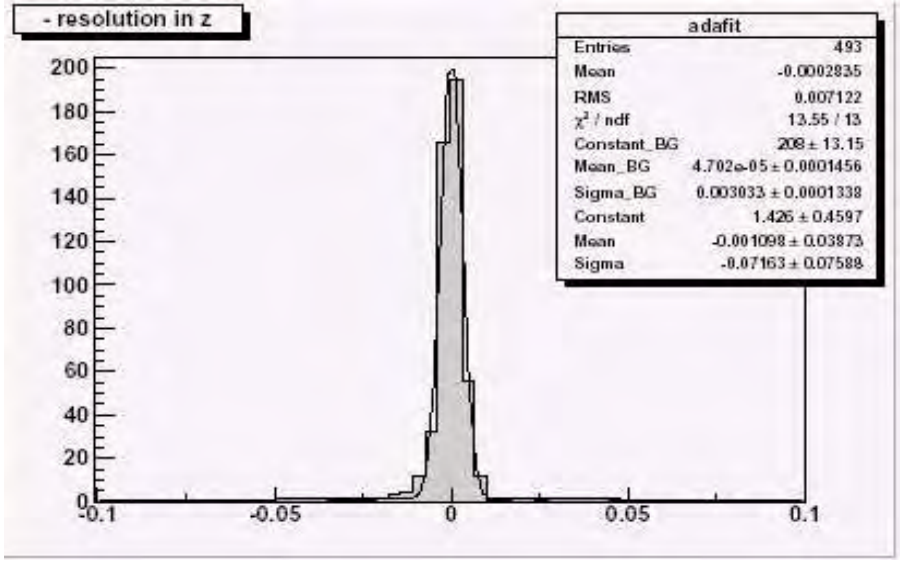
The Apex Point formalism may also be used as a **linearization point finder**.

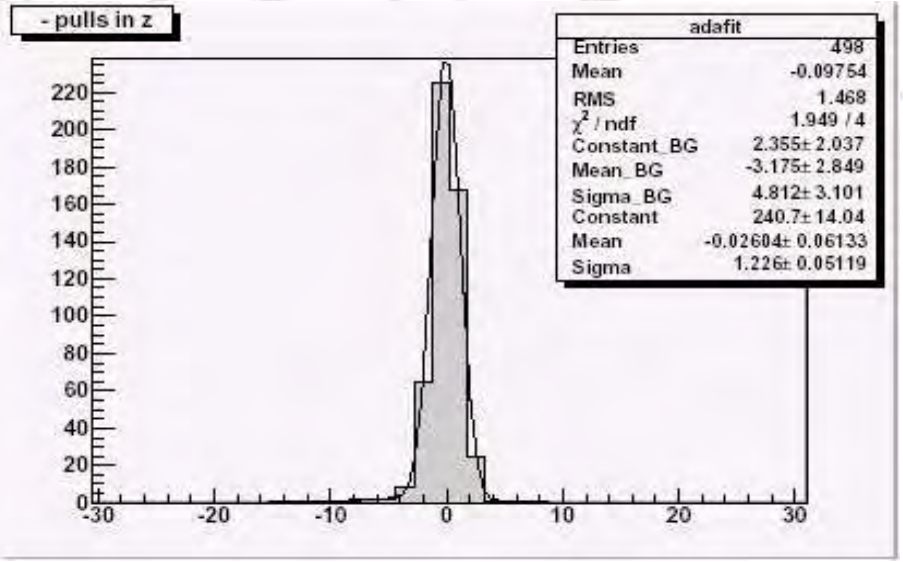
AdaptiveVertexFitter

$$\hat{\beta}_{Adaptive} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{n} \left(w_i \cdot r_i^2(\beta) \right)$$

- fast iterative, re-weighted LS fit with annealing.
- breakdown point: 0.5
- weight $w_i(r_i)$ of track i at iteration k depends on distance r_i to vertex at iteration k-1, and temperature
  - $w_i(r_i) \equiv$ assignment probability
  - r_i = reduced distance
- general-purpose algorithm
- user can choose cutoff on r_i and annealing schedule

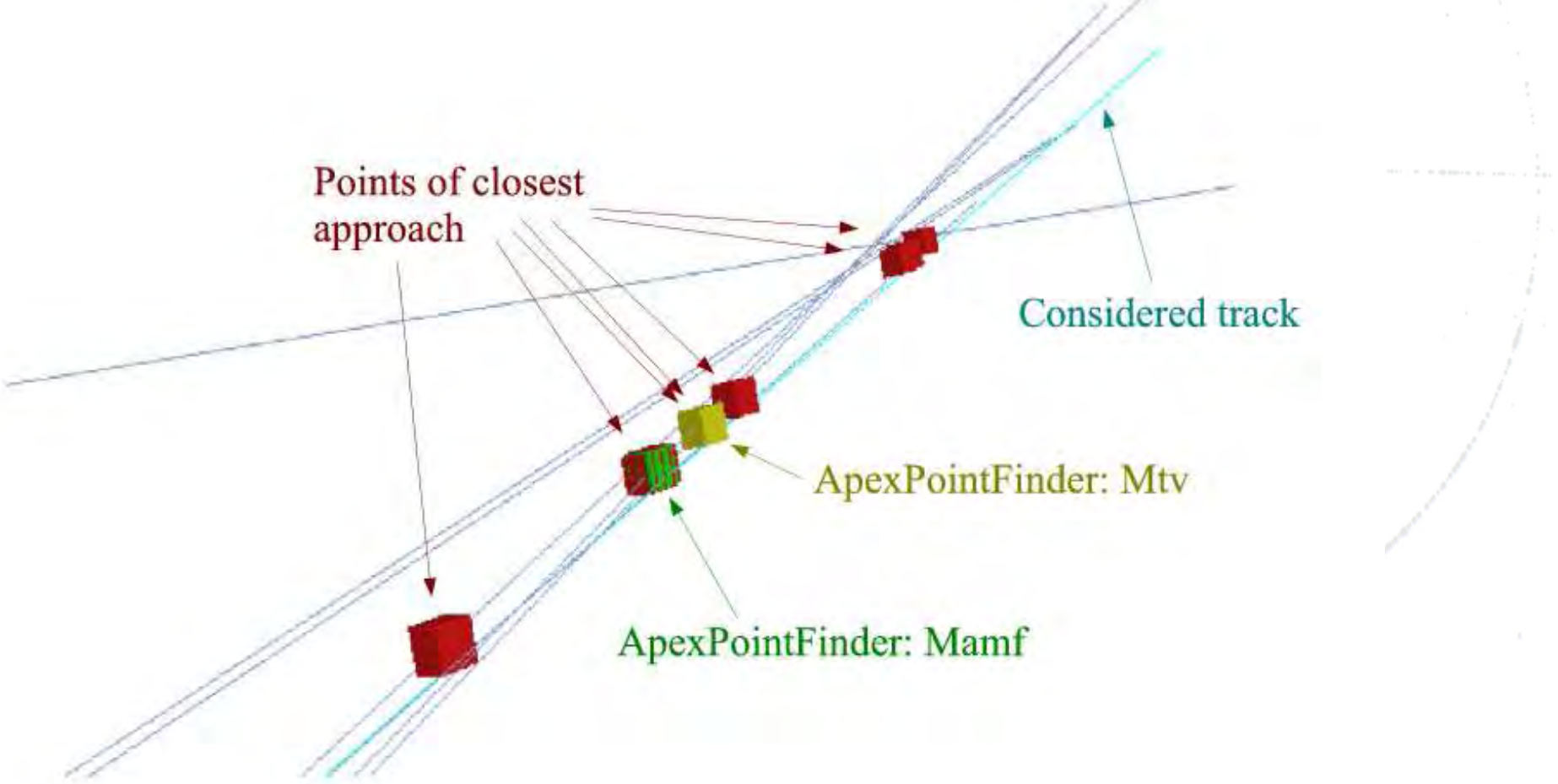






Apex Points

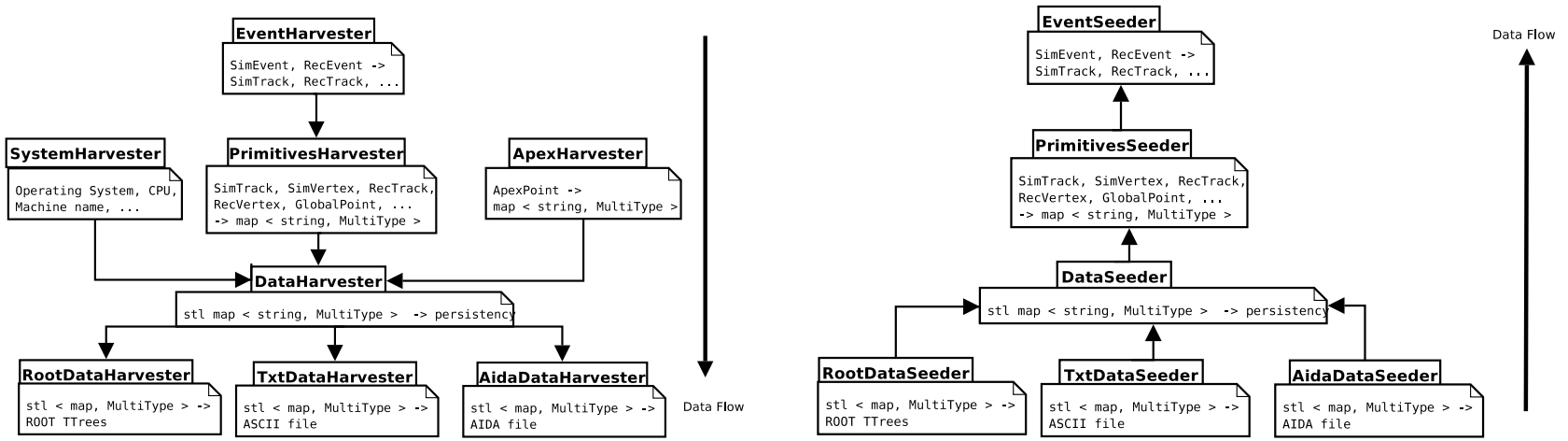
To fix the problem with the triangle inequality, one may try to find **a point that fully represents the track**. One such point could be the ApexPoint; that is a cluster point in the set of all Points of Closest Approach that lie on the considered track. The most promising ApexPoint finding algorithm is "Mtv": Minimal Two Values.



Analysis & debugging tools (1)

Persistency storage solution

- A persistency storage solution was originally realized on top of ROOT; it is currently being extended by more standard-compliant alternatives (AIDA and XML). `DataSources` include a "vertex gun", LCIO, etc.
- All I/O is handled through a "data harvesting" concept (which may possibly be integrated as front/end in AIDA): object → STL map → ASCII/ROOT/AIDA file (`Harvester`) and vice versa (`Seeder`).
- The STL mapping is heterogeneous: it handles int/double/string objects as `MultiType`.



Analysis & debugging tools (2)

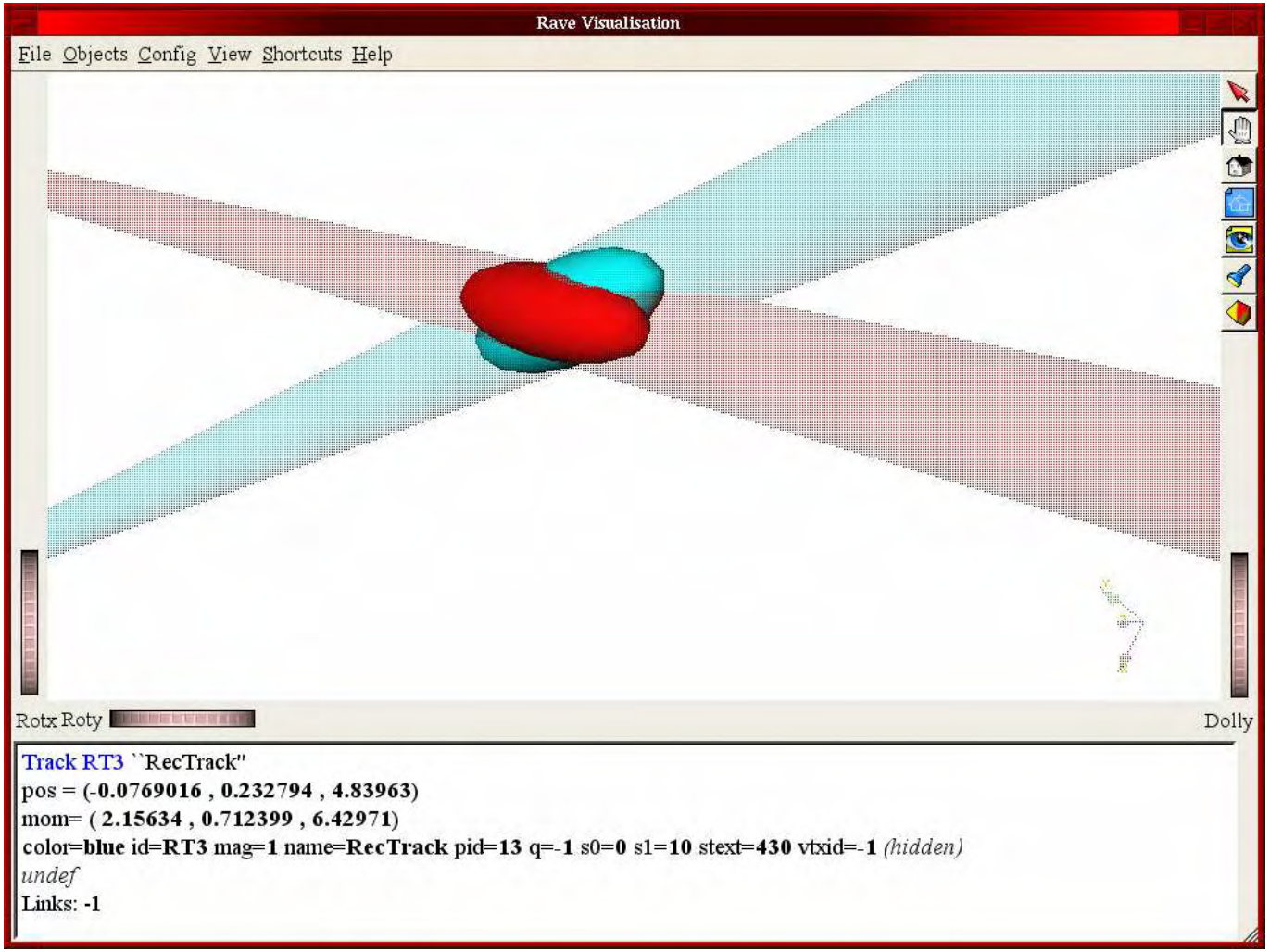
Visualisation tool

- Visualisation is deliberately kept simple for the sake of detector-independence. It follows the model-view-controller (MVC) paradigm and is based on COIN3D.
- Object data are accessed as `MultiType` STL maps:
  - at present only indirectly from an ASCII/ROOT/AIDA file through the `Seeder`;
  - in future maybe also directly through the `Harvester` and a TCP stream.
- Interactivity is at present limited to manipulators on graphic objects. The tool may later be augmented with full-scale interactivity, to be provided by PYTHON (or some other scripting language).

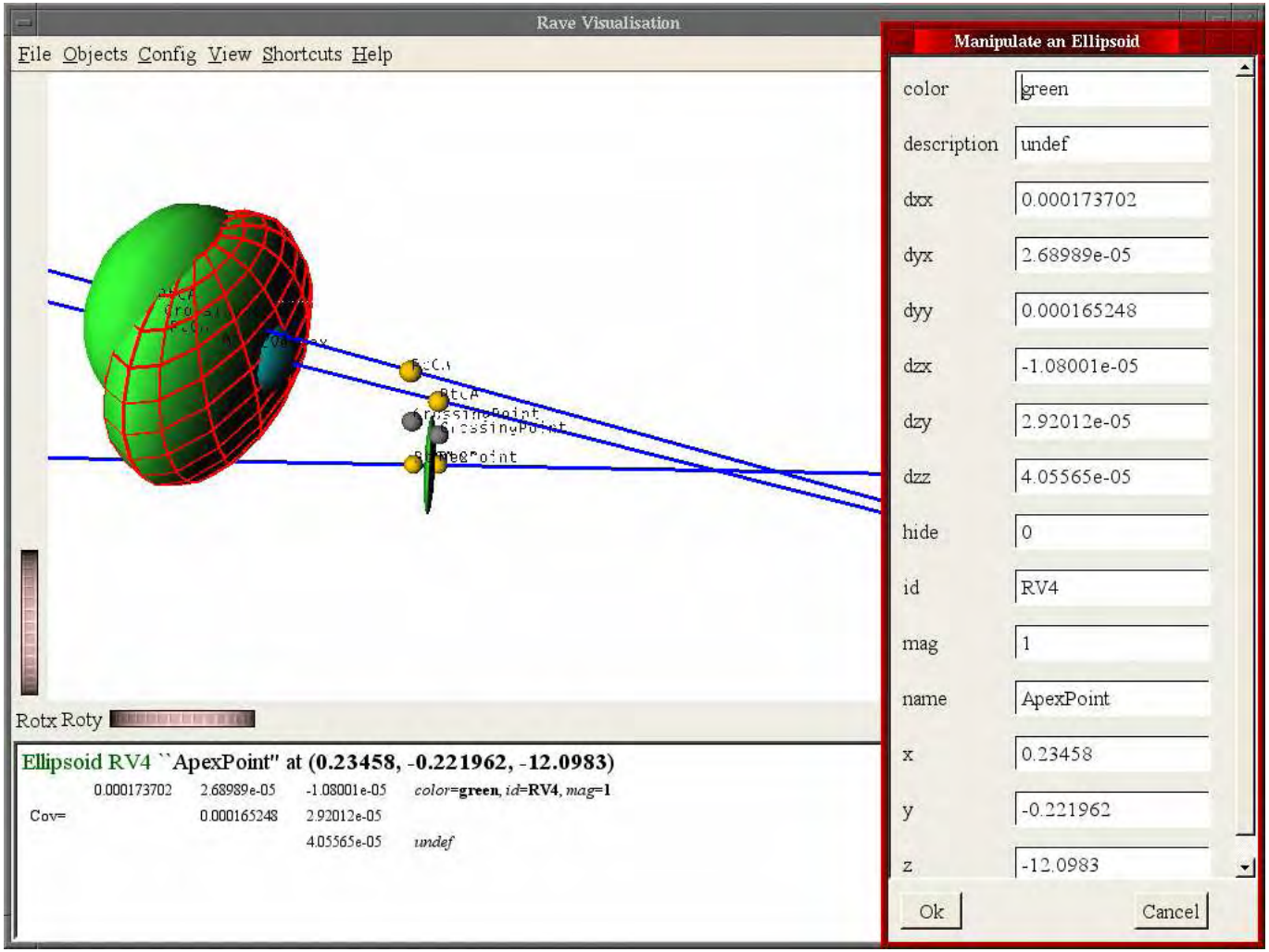
Example snapshots

- **Example 1:**
  - Two reconstructed tracks (with their transversal errors shown as a tube), together with their apex points (with error ellipsoids).
  - The parameters of one track are displayed at the bottom.
- **Example 2:**
  - Three reconstructed tracks, together with two (out of three) triples of points of closest approach (yellow) and crossing points (gray); with their three apex points (with error ellipsoids, green); and with the fitted vertex position (with error ellipsoid, blue).
  - The parameters and covariant errors of one apex point are displayed at the bottom. The corresponding manipulator window is displayed on the right.

Visualisation tool – example 1



Visualisation tool – example 2



Analysis & debugging tools (3)

Math library package

- The choice of `MathPackages` (including linear algebra) is crucial for the efficiency and reliability of the toolkit.
- CLHEP appears to be the only choice freely available today, but there are serious doubts about its reliability.
- NAGlib is a reliable alternative, but may be too expensive for users outside of campus licence agreements.
- Generic (template) libraries would be our preferred choice. Candidates exist, e.g. MTL, GSL, FLENS, SLATE, BLITZ (all GPL-licenced); but none providing the full functionality required.

The SKIN concept

- Different experiments will use different sets of the optional packages. There are two alternatives:
  - the package is part of and is shipped with VERTIGO; or
  - the package is maintained by the particular experiment, and VERTIGO provides only the appropriate interface.
- An experiment-specific set of packages is called a SKIN. Examples are a stand-alone skin (called the "Framework"), CMS skin, ATLAS skin, TESLA skin, LCD skin, etc.
- Pre-defined skins may easily be selected by the user with `configure`.
- Maintenance and distribution of the toolkit is supported by a CVS repository at HEPHY Vienna.

Conclusions and outlook

Impact of the project

- This is (according to our knowledge) among the first large-scale attempts of refining a substantial part of reconstruction software into a detector-independent toolkit.
  - The idea of a detector-independent vertex library, based on the robust M-estimator and written in FORTRAN, was proposed by one author *(WM)* already in 1996.
  - Nowadays, this project is supported by recent advances in information technology, together with an overall increased IT maturity of the HEP user community.
- Interests in using the toolkit, once it will be released, have been expressed by CMS, ATLAS, LHCb, BELLE and Linear Collider (TESLA, LCD) collaborators. More to follow ?
- For track reconstruction, a general-purpose toolkit (RecPack) is currently being developed, following similar ideas, by *A. Cervera-Villanueva* et al. Possible synergies with our project (e.g. by data sharing, use of common interfaces, etc) are welcome and are actively pursued.

Manpower and funding

- So far, the project has been pushed by the enthusiasm of one author *(WW)*, with some help from a part-time graduate student, at HEPHY Vienna. External funding for a postdoc will be applied for.
- **Closer collaboration among the contributing labs is welcome and will be essential for success.**