DISSERTATION

Development of Vertex Finding and Vertex Fitting Algorithms for CMS

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Doktors der technischen Wissenschaften unter der Leitung von

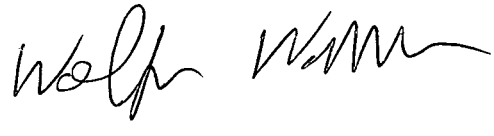
Univ. Doz. Dipl.-Ing. Dr. Rudolf Frühwirth (E107)
am Institut für Hochenergiephysik der Österreichischen Akademie der Wissenschaften

eingereicht an der Technischen Universität Wien
Fakultät für Physik

von

Dipl.-Ing. Wolfgang Waltenberger
E9325377
Gumplowiczstraße 1/4/21
A-1220 Wien
walten@hephy.oeaw.ac.at

Wien, am 19. Oktober 2004



Kurzfassung

Im CMS Experiment müssen Interaktionswechselwirkungspunkte ("Vertices") der Teilchen rekonstruiert werden. Man unterscheidet zwischen "primären" Proton-Proton Kollisionsvertices und "sekundären" Zerfallsvertices. Das Problem der Vertexrekonstruktion wird des weiteren aufgeteilt in das statistische Problem, einen Vertex aus einer gegebenen Menge an Teilchenspuren zu schätzen, und in ein Mustererkennungsproblem. Hierbei ist die Aufgabenstellung, aus einer Gesamtmenge Untermengen von Teilchenspuren zu finden, die einen gemeinsamen Ursprungsvertex teilen. Die vorliegende Arbeit beschäftigt sich mit allen oben genannten Aspekten. Im Gebiet der Vertexschätzung wurden Robustifizierungen des Kalman Filters entwickelt, implementiert und untersucht. Die Vertexsuche betreffend wurden verschiedene, teils aus der Literatur bekannte Algorithmen der Problemstellung angepasst, implementiert und analysiert. Zu diesem Zweck musste eine Reihe an Softwaretools geschrieben werden. Es war notwendig, dem Anwender volle Kontrolle über die kinematischen und statistischen Eigenschaften zu ermöglichen. Dafür wurde ein Paket zur schnellen Simulation entwickelt. Eventweises Debuggen hingegen führte zur Entwicklung eines Visualisierungstoolkits. Ein einfacher Persistenzmechanismus ("data harvesting") ermöglichte schnelleres Debuggen. Erst ein Paket, das Vertexsucher automatisch fein abstimmt, erlaubte einen unverzerrten Vergleich der verschiedenen Algorithmen.

Alle Algorithmen wurden auch in realistischen Anwendungen getestet. Im Gebiet der Vertexsuche wurde das Hauptaugenmerk auf die Verwendung der Vertexsucher im "*b*-tagging" gelegt. Im Bereich der Vertexschätzung wurde das Schätzen von Primärvertices mit Ausreißern und Sekundärvertices mit schlecht gemessenen Spuren getestet. Es konnte gezeigt werden, dass der adaptive Schätzer den klassischen Methoden, die mit "harter" Zuordnung arbeiten, überlegen ist. Der Multi-Vertexschätzer konnte im Vergleich zu den klassischen Strategien bei gleichbleibenden *b*-tagging Ergebnissen die *b*-mis-tagging Raten wesentlich verringern. Darüberhinaus wurde ein exzellenter Kandidat für die Online-Vertexrekonstruktion gefunden: der adaptive Vertexrekonstruktor erzielt ebenfalls gute *b*-tagging Ergebnisse, und ist außerdem sehr schnell.

Robuste Schätzer sind in großem Maße abhängig von ihrer Initialisierung. Verschiedene "linearization point finders" wurden analysiert. Ein bestimmter Algorithmus wurde als neuer Standardalgorithmus festgelegt. Er ist sehr schnell und liefert in einem weiten physikalischen Anwendungsbereich ausgezeichnete Anfangswerte. Viele "linearization point finders", darunter auch der neue Standardalgorithmus, bedürfen einer schnellen und dennoch präzisen Methode, den minimalen Abstand zwischen zwei Helices zu berechnen. Ein neuer Algorithmus wurde implementiert; er ist den bisherigen Methoden weitaus überlegen.

Abstract

In the CMS detector interaction vertices of particles will have to be reconstructed. We distinguish between proton-proton collision vertices ("primary vertices") and decay vertices ("secondary vertices"). We furthermore categorize into fitting a given set of tracks into a vertex, and finding sets of compatible track bundles, "vertex finding". This PhD thesis deals with the tasks of finding and fitting of primary and secondary vertices. In the field of vertex fitting robustifications to the classical Kalman filter formalism have been developed, implemented, and tested. In vertex finding several algorithms found in literature have been identified, implemented, and tested. To this end it was necessary to develop a fairly large set of tools. The necessity for full control over event kinematics and the reconstructed tracks lead to the implementation of a fast simulation package. The event-by-event debugging process triggered the development for a special-purpose visualization. A simple persistency mechanism ("data harvesting") that allows for faster debugging cycles was developed. A package that automatically tunes vertex finders made fair comparisons between different algorithms possible.

All algorithms have also been tested in realistic use cases. On the finding side strong emphasis was put on the use case of employing vertex finding algorithms in the context of b-tagging; vertex fitting was tested in the application of primary vertex fitting with outlying tracks and secondary vertex fitting with mis-measured tracks. It could be shown that the adaptive vertex fitting method is superior to the classical algorithms that operate with "hard assignment". In the vertex finding section, the multi vertex fitter method improved b-mis-tagging rates significantly, while maintaining the same level of b-tagging rates as the classical strategies. Also, a likely candidate for online vertex reconstruction could be identified: the adaptive vertex reconstructor. This method, while it is extremely fast, also exhibits an excellent b-tagging performance.

Robust fitting methods depend heavily on their initialization. Various linearization point finders were studied, one of which was identified as the new default algorithm. It is a fast algorithm which has an excellent performance over a wide range of applications. Many linearization point finders, including the default method, need good and fast ways to compute the minimal distance between two helices. A new algorithm was implemented that outperforms the previous methods.

Acknowledgment

First of all my thanks go to my supervisor Rudi Frühwirth. Both Rudi's expertise in "his" field and his social skills that are, it seems, just as crucial for a successful supervision of a PhD thesis as the scientific competence, are widely known and well documented [104, 74, 92]; every additional comment seems quite redundant. So let me be indeed be redundant: it has been a truly grand experience working with Rudi. Further thanks for supervision to Pascal Vanlaer, my *de facto* second supervisor, for inserting me into the ORCA vertexing group. Pascal was a knowledgeable source of information, who, amazingly, always had an open ear for all silly and not-so-silly requests that one has in three years of writing a PhD thesis.

Further thanks goes to all my colleagues both in Vienna and in Geneva, Meinhard Regler, Winni Mitaroff, Laurenz Widhalm. Special kudos to Gerald who has shared the subtle joys of the open source world with me (predominantly the CLI parts of it, since our opinions on the GUI world diverge), Kirill Prokofiev for a few wonderful evenings in quite a variety of European cities, Christian Weber for periodically inserting me into the miraculous world of phenomenology and "how to get lost in the particle zoo". My last collegial thanks go to Susanna, friend and moral support in an environment as bizarre as the CERN environment can be at times.

Last but not least all my thanks to my dear family – Manja who has, herself, had a thesis to write [105], Fabian who in the same period in which I managed to write but one little PhD thesis, realized great many "first ones": he made his first steps, uttered his first words, played his first games, asked his first questions, watched his first movies, and, ultimately, logged onto our Linux computer for the first time in his life! This nicely documents what a slow and tedious process science generally is, or, for once, how incredibly adaptive children are. Finally special thanks also to my parents. I doubt there is anybody who knows me since longer.

Parts of the LaTeX style of this thesis are taken from the source code of R.M. Hristev's "Matrix ANN" book [65].

Further thanks also goes to "my students" (note the *composite* nature of being a student), Boris Legradic, for the templated version of the voting schema, David Schmitt, for his implementation of the vector quantization algorithm (and a few smart ideas about quite a few topics), and Florian Köchl, for the implementation of the Super Paramagnetic algorithm, and Thomas Layer and Werner Prinot who have very recently started to work on the PickleHarvester.

Plenty of world-readable afs disc space (50GB) has been granted to the author for the production of Monte Carlo data. Thanks a lot to Gerhard Walzel and his computing team at the HEPHY for this generous gesture.

This thesis was made possible by the Fonds zur Förderung der wissenschaftlichen Forschung, project number 15177.

Contents

The White Rabbit put on his spectacles. "Where shall I begin, please your Majesty?" he asked. "Begin at the beginning,", the King said, very gravely, "and go on till you come to the end: then stop."

Lewis Carroll, "Alice in Wonderland"

CHAPTER 1

Introduction

Nos esse quasi nanos gigantum umeris insidentes, ut possimus plura eis et remotiora videre, non utique proprii visus acumine aut eminentia corporis, sed quia in altum subvehimur et extollimur magnitudine gigantea.
(We are dwarves perched on the shoulders of giants. We see thus more and further than they do, not because our sight is more acute or our height taller, but because they lift us into the air and elevate us with all their gigantic height.)

Bernard de Chartres

Introductions commonly have it that they provide a starting point for the story that is being told. This document shall not deviate from this rule. It shall be attempted to bring to the eye of the reader the greater picture that this work is embedded in. Starting with a description of the physics goals of LHC, the hardware setup and a few details of the CMS inner tracker and the calorimetry will be given. The data challenges that have to be met shall then be presented, and the time budget available for vertexing within a high level trigger will be given. The chapter shall end with a brief summary of the contribution of this PhD thesis to the CMS experiment.

1.1 Physics at the LHC

In Nature's infinite book of secrecy a little can I read.

William Shakespeare, "Anthony and Cleopatra"

To say that the experiments at the Large Hadron Collider (LHC) [31, 8] will dominate the next decade of high energy physics is probably not exaggerated. The range of physics models LHC experiments will cover, the number of theories that will be at test, is large. Speaking in simple terms, the following qualitative, "epic" questions are expected to be answered:

- What is the origin of mass? Within the standard model all particles acquire their masses by coupling to the (physically observable) Higgs boson, but the latter has not yet been observed.
- Is there experimental evidence for a Grand Unified Theory (GUT) – a proposed unification of the electromagnetic, the weak, and the strong interaction?
- Is a fundamental symmetry between fermions and bosons, the "supersymmetry", realized in nature?
- What is the origin of "dark matter"? Could the so-called neutralinos, particles proposed by the theory of supersymmetry, be a candidate for dark matter?
- Why is there an asymmetry in the quantity of matter as compared to antimatter?
- Why are there three families of quarks and leptons?
- Do quarks and leptons have a substructure?
- Is there a new form of matter, the so-called quark-gluon plasma, as it might have existed in the early universe?
- Is the world we live in really four-dimensional? Will evidence for further dimensions be found?

The following sub-sections shall briefly delve into a few main potential discoveries, focussing on possible contributions of this thesis.

1.1.1 The Higgs boson

(The deep of night is crept upon our talk,) And Nature must obey necessity.

William Shakespeare, "Julius Cæsar"

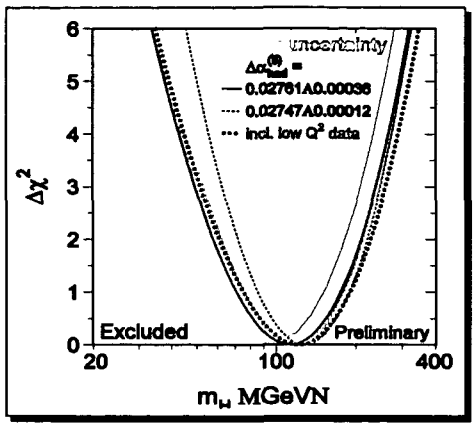


FIGURE 1.1: $\Delta\chi^2$ of the electro-weak "world" fit, as a function of the Higgs mass

The standard model today is left with one and only one piece that still defies experimental verification – the Higgs boson [90, 97]. This Higgs field was introduced into the model as a mechanism that allows insertion of mass terms in the Lagrangians without explicitly violating the fundamental gauge symmetries – the $U(1)_Y$ and the $SU(2)_L$ symmetries of the electro-weak theory. In its simplest form it is inserted as a doublet field, with $2 \cdot 2 - 3 = 1$ degrees of freedom, that is, with one physical particle. The theory itself does not predict the mass of the Higgs boson. But it *does* predict how the mass depends on the branching ratios. The past LEP experiments have not found the Higgs boson, but they have left us with a lower limit on the mass: $M_H > 114.4$ GeV at 95% confidence level, see Fig. 1.1.

Higgs boson production

The leading process for Higgs boson production over the whole mass spectrum is gluon fusion with no by-products, see Fig. 1.2. The production rate of the next-to-leading production mechanisms, like vector boson fusion ($qq \rightarrow Hqq$) or W/Z bremsstrahlung ($qq \rightarrow HW/Z$) are, in a great part of the spectrum, already orders of magnitude smaller than the gluon fusion. The only other production channel that is of relevance for this thesis, is the $t\bar{t}H$ channel; this potential discovery channel has a much lower branching ratio, but the decay products of the Higgs boson and the top quarks make for a very distinct signature.

The most prominent decay channels for standard model Higgs particles are as follows:

- For $M_H < 140$ GeV (small mass scenario): The decay channel with the highest branching ratio in this scenario is $H \rightarrow b\bar{b}$. A large QCD background, unfortunately, makes it unusable for physics analysis. Thus, one of the promising decay channels is $H \rightarrow \gamma\gamma$; the narrow width of the Higgs boson and the expected superb mass resolution should allow to discriminate the signal from an immense background (note the very low branching ratio $\sim 10^{-3}$). One possible alternative in this low mass region

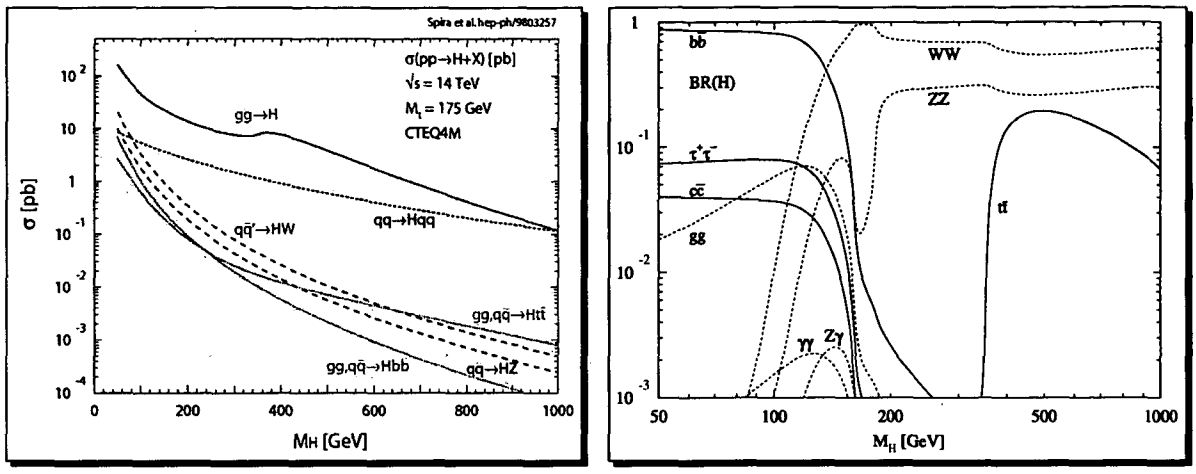


FIGURE 1.2: Higgs production channels (left), Higgs decay channels (right), taken from [90]

is to look at Higgs bosons with other particles as by-products, e.g. Higgs bosons with an associated $t\bar{t}$ pair, Fig. 1.3. In this case one can trigger on e.g. the leptonic decay of one of the top quarks. The event has $4b$-jets; good b-tagging capabilities will indeed be crucial in this complicated channel.

- For $130 < M_H < 500$ GeV (medium mass range): The most prominent channels are $H \rightarrow VV^* \rightarrow l^+l^-\nu\nu$, and $H \rightarrow VV^* \rightarrow l^+l^-l^+l^-(l = e, \mu; V = W, Z)$. These channels are fully leptonic. Especially the four-lepton channel has a particularly clean signature; it is sometimes referred to as the gold-plated channel.

- For heavy Higgs ($M_H > 500$ GeV): For this mass range $H \rightarrow VV^* \rightarrow l^+l^-$jet jet is one of the more promising channels.

The LHC accelerator has been specifically designed to cover the whole Higgs mass range up to about 1 TeV. Within this range, the Higgs particle can be expected to be found within the first year of running at low luminosity.

Finally note that the predicted Higgs particle is a scalar, but there is nothing in the theory that demands that it be a fundamental scalar field: Higgs "condensates" are a possibility.

1.1.2 Supersymmetry, SUSY Higgses, and SParticles

The best argument for supersymmetry: almost half of the predicted particles have already been found.

Supersymmetry is a proposed extension to the standard model that combines fermionic and bosonic states. Its theoretical achievements include the unification of all fundamental

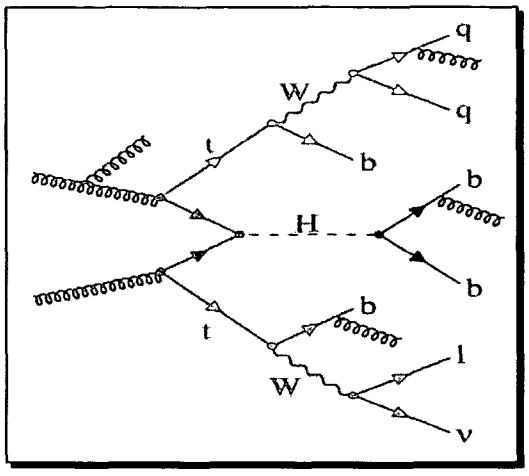


FIGURE 1.3: Feynman graph of the ttH channel; a Higgs boson with an associated top quark pair production.

forces (except for gravity) at the GUT scale, and the cancellation of all quadratic divergences in the QFT perturbation theory. Its smallest possible form is referred to as the minimal supersymmetric model (MSSM). It introduces a supersymmetric partner for every particle; the Higgs doublet is extended to comprise two doublets, making way for no less than $2 \cdot 2 \cdot 2 - 3 = 5$ physical MSSM Higgses: a CP-odd (A), two CP-even (H, h), and two charged ($H^{\pm}$) Higgs bosons. The spectrum of the MSSM Higgs bosons can at tree level be expressed with only two parameters; m_A and $\tan\beta$. The predicted phenomenology, though, depends also on the SUSY particle masses w.r.t. the Higgs particle masses. If the SUSY particles are heavier than the Higgs bosons, then evidently the latter cannot have supersymmetric decay chains. So, even for the MSSM the spectrum of possible phenomenology is extremely wide; it makes little sense to discuss specific scenarios here. But it *can* still be said on this abstract level that b-jets will play an important role either as part of a signal or as part of background. For a good overview over supersymmetry and its predicted phenomena see [78].

SParticles and particlinos

The predicted supersymmetric particles are the

- squarks ($\tilde{q}$), sleptons($\tilde{l}$) and sneutrinos($\tilde{\nu}$) - the bosonic supersymmetric partners of the quarks, electron, the muon, the τ lepton, and the neutrinos, respectively.

- gluinos($\tilde{g}$), charginos($\tilde{\chi}^{\pm}$), and neutralinos($\tilde{\chi}^{0}$), the supersymmetric partners of the gluons, the gauge bosons and the Higgs boson. (The charginos and neutralinos are superpositions of the "mathematical" gauginos and Higgsinos). In many scenarios the lightest SUSY particle is a neutralino. This particle decays no further, if R

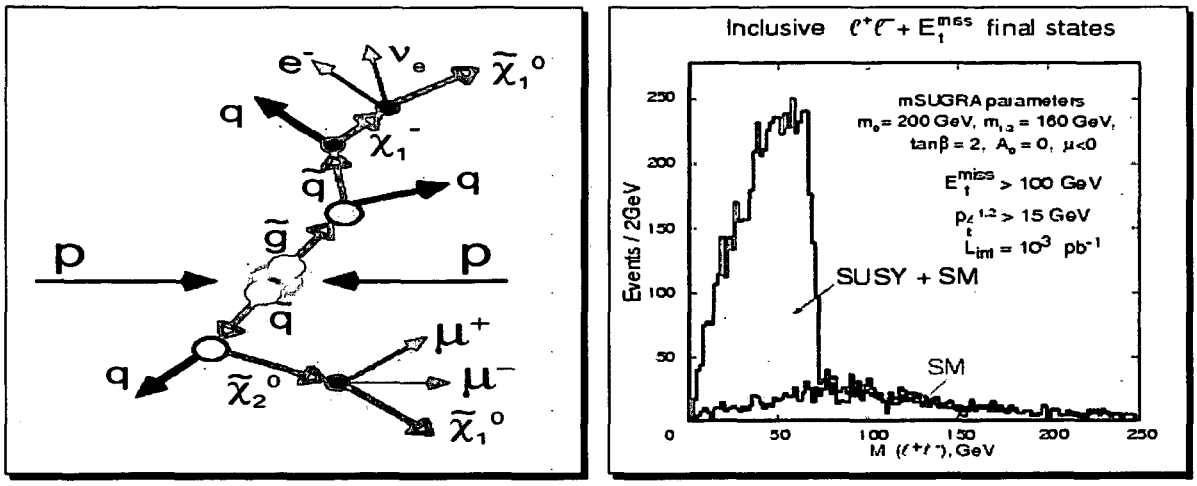


FIGURE 1.4: Left: A SUSY decay chain (with quite a spectacular signature). Right: The di-lepton mass distribution, after one month of data taking at low luminosity. (Plots taken from [99])

parity is conserved, i.e. if the number of SUSY particles is conserved. In this case it is a stable particle — a weakly interactive massive particle (WIMP). It makes an excellent candidate for dark matter (see [78] and references therein).

Note that the question of whether or not supersymmetry is realized in nature will probably be answered very quickly, once LHC is running. See Fig. 1.4 for an example of how one particular decay chain (left plot) can dramatically change the (inclusive) di-lepton mass distribution (right) after one month of data taking [62].

1.1.3 Large extra dimensions and black holes

(Wherefore base? When) my dimensions are as well compact, (My mind as generous, and my shape as true as honest madam's issue?)

William Shakespeare, "King Lear"

If the Planck scale is in the TeV range, LHC will produce one black hole about every second [44, 63] [1]. The microscopic black holes will in this scenario decay into prompt, hard photons and energetic leptons which, together, will provide a very clean signature. They will thus be easily detectable. Apart from the first direct confirmation of Hawking radiation, information could be revealed about the scale of quantum gravity, and the number of (large) extra dimensions.

[1] There are models that predict a lower production rate even with a $\mathcal{O}$(TeV)-Planck scale; literature has at least two candidate mechanisms for suppression of black hole production: Voloshin suppression [81] and the generalized uncertainty principle [30].

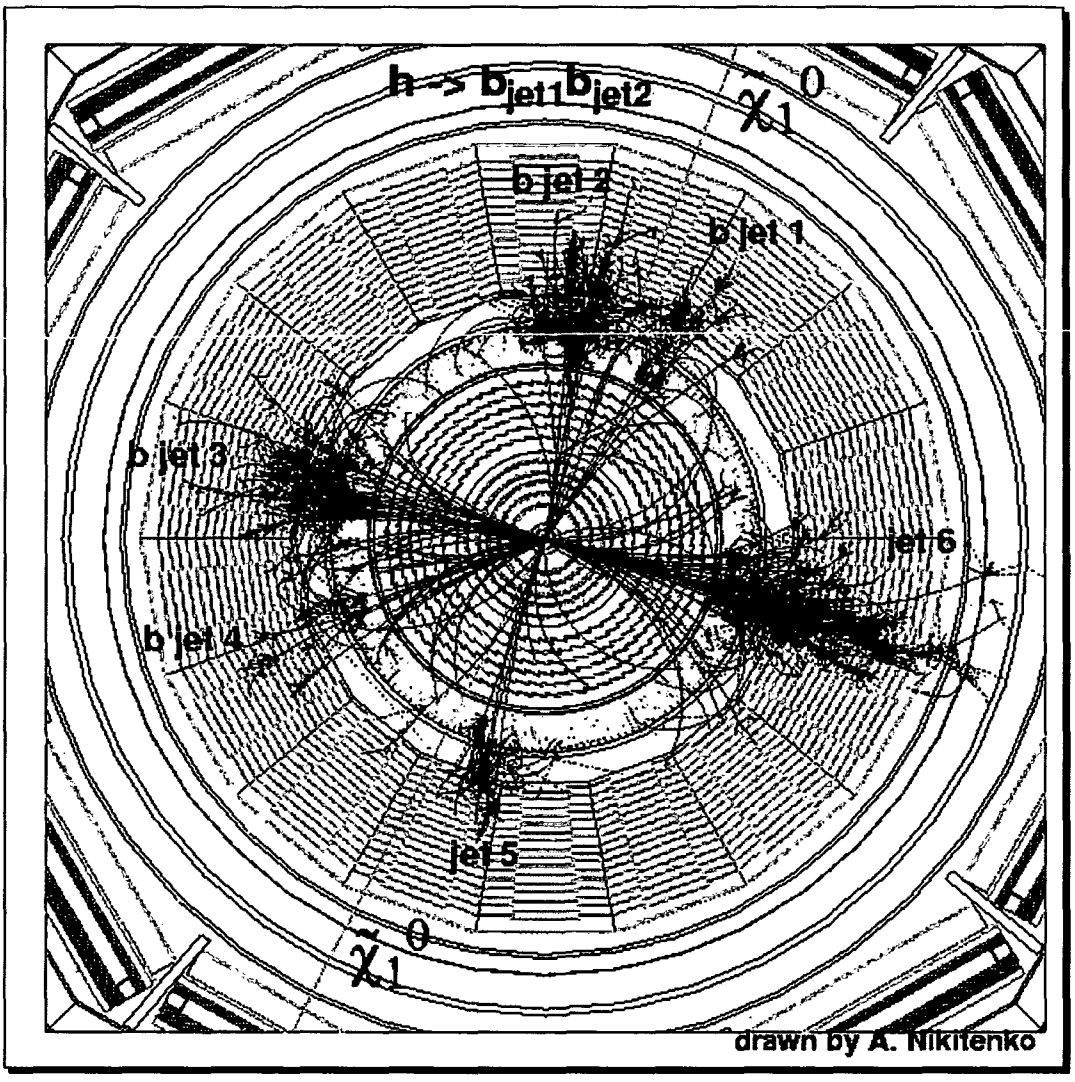


FIGURE 1.5: A SUSY Higgs $\rightarrow b\bar{b}$ decay, taken from [3]. This is a CMSIM simulation of an mSUGRA event. The pp-collision produced $\tilde{u}_L + \tilde{g}$. $\tilde{g}$ decays in a lengthy chain into jets 3, 4, and 5. $\tilde{u}_L$ decays into a u-quark (jet 6) plus a neutralino and a SUSY Higgs h. The Higgs h then decays into $b\bar{b}$ that generates the jets 1 and 2.

1.1.4 Exotics

You should never bet against anything in science at odds of more than about 10^{12} *to* 1.

Ernest Rutherford

The number of more daring ideas that can and will be tested in LHC is large. It is futile to even start discussing possible discovery channels, let alone discussing potential use cases of sophisticated vertex reconstruction methods. Hence we shall restrict ourselves to simply listing a few physics models:

- **String balls** [45, 32] Highly excited strings can appear at an energy range similar to black holes – in models that assume a $\mathcal{O}$(TeV)-Planck scale.
- **Little Higgses** This model [23] introduces a Higgs boson as a pseudo-Goldstone boson resulting from a spontaneously broken global symmetry. The search for little Higgses is pursued by one ATLAS group [58].
- **Leptoquarks** Leptoquarks are hypothetical particles predicted by a wide range of models. Living in "both worlds", they carry a color charge, a fractional electric charge, and both baryon and lepton numbers.
- **Extended gauge groups** (Z') All GUTs "above" SU(5) — the simplest possible group which can contain the standard model — predict at least one extra neutral gauge boson [73]. The discovery of such a Z' boson would provide information on the GUT group and its symmetry breaking.
- **Randall-Sundrum models** [80] Randall-Sundrum models postulate a five-dimensional Anti-de-Sitter space-time with two 4d branes, which are commonly referred to as the "Planck" brane and the "standard model" brane, respectively. The warped (small) extra dimension is a possible answer to the hierarchy problem.

1.1.5 b-tagging

Der kleine Gott[2] *[...] in jeden Quark begräbt er seine Nase.*

Johann Wolfgang von Goethe, "Faust"

The identification of b-quark decays has already played important roles in past experiments; more recent examples are the $Z \to b\bar{b}$ branching ratio at LEP, and the discovery of the top quark with $t \to bW$ at the Tevatron. In the LHC experiments b-tagging will very likely play a crucial role in many analyses, be it for signal discrimination (as in e.g. the ttH channel), be it for background subtraction as it will happen in a lot of channels.

[2] "Der kleine Gott" is a reference to man.

The average lifetime of a B meson is $c\tau_{B^0} \sim 464\mu$m. Considering the Lorentz boost ($\gamma \sim 4-5$), this results in a flight path in the detector of about two millimeters. B mesons produce on average five charged particles per decay. It is the task of vertex reconstruction in the b-tagging context to find this secondary vertex; b-tagging algorithms can exploit the information of the existence of a secondary vertex, together with the number of tracks associated to the vertex, and the total track momentum at the vertex. This can be used to separate B mesons from its background, like the decay of a D meson, which has a considerably shorter life time ($c\tau_{D^0} \sim 124\mu$m), and fewer tracks and a smaller mass at the vertex. The information obtained by vertex reconstruction is not the only source of information of a b-tagger; another possibility for a b-tagger is to simply count the number of tracks above a certain threshold on the impact parameter significance [86].

1.2 The CMS Detector

The Compact Muon Solenoid (CMS) detector [39, 36] is one of the four detectors at the future Large Hadron Collider (LHC). LHC will produce proton-proton collisions at unprecedented energies: The total center-of-mass energy will be 14 TeV. This will produce parton-parton interactions at the scale of up to a few TeV. LHC experiments generally and CMS in specific are designed to answer a large range of open fundamental questions of high energy physics. The basic design criteria are as follows:

(a) a very good identification and momentum measurement of muons ($\mu^{\pm}$),

(b) energy measurement of photons (γ) and electrons/positrons ($e^{\pm}$) with best possible precision and consistent with (a),

(c) a central tracking detector for precise momentum and impact parameter measurements of charged particle tracks to facilitate (a) and (b),

(d) robustness and cost effectiveness while maintaining (a), (b), and (c).

Fig. 1.6 shows a schematic view of the detector. A collaboration of nearly 2000 scientists and engineers from more than 30 nations and over 150 scientific institutions is currently designing and building the detector which is supposed to go online in 2007. The next sub-sections will briefly walk through the detector sub-systems. The layout of the detector is subject to changes — e.g. in the first year of operation CMS is known to start in a simplified setup. Reports and up-to-date information can always be queried from the CMS technical web pages [35].

1.2.1 The CMS inner tracker

The tracker [40, 34] constitutes the central part of the CMS detector. It is a silicon detector, consisting of 2d "pixel" detectors and one-dimensional silicon microstrips. It

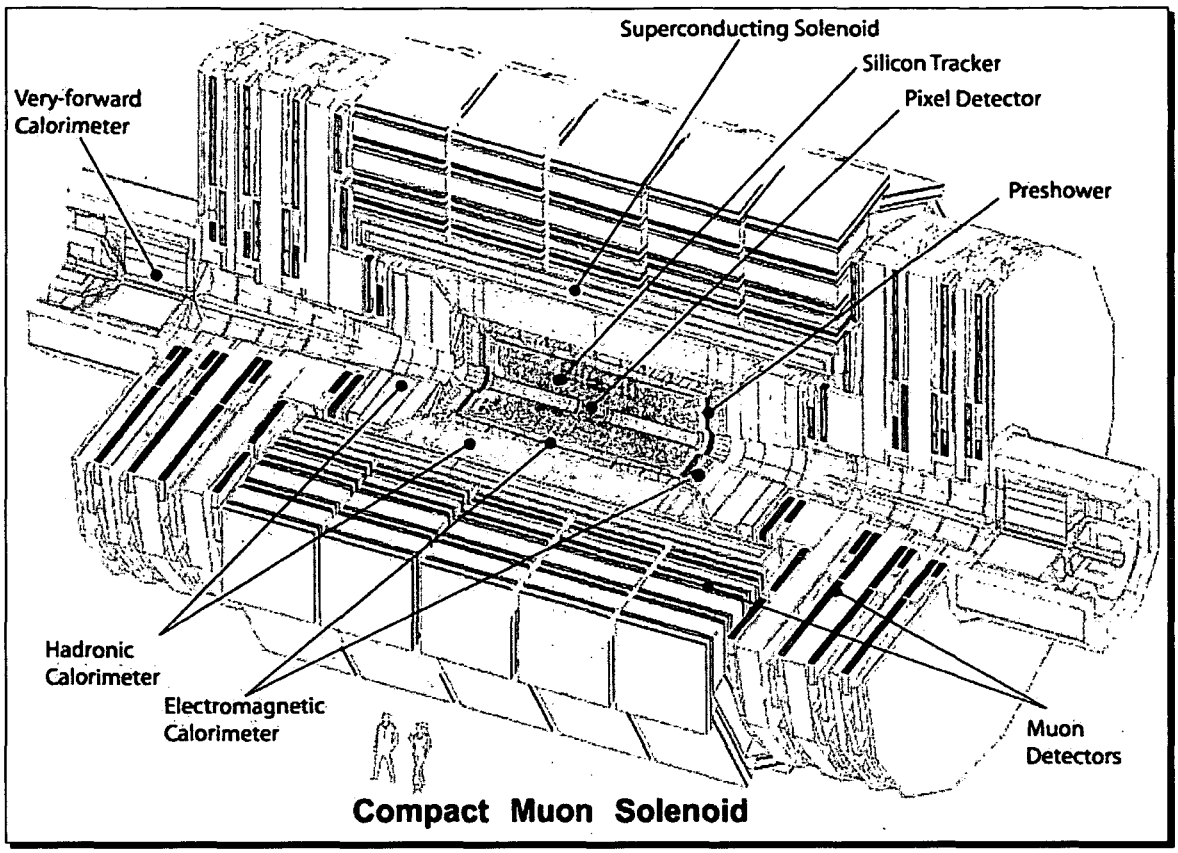


FIGURE 1.6: The CMS detector and its components. We can see the large muon chambers, the hadronic and the electromagnetic calorimeters, and the "inner tracker" – consisting of the silicon microstrip detectors and the "pixels".

measures (i.e. it "tracks" the flight paths) of charged particles. When crossing materials, charged particles lose energy, mainly due to ionization. This effect of ionization is measured and transformed into electrical signals by dedicated read-out hardware. In order to sustain the enormous particle flux (luminosity) of the LHC, the following requirements have to be met:

- the detectors need to be radiation hard,
- their response needs to be fast (one bunch-crossing every 25ns),
- the number of channels per sensitive element needs to be sufficiently high in order to achieve a low enough occupancy (of the order of a few percent),
- The spatial granularity needs to be fine enough for a high precision of the measurement of the track parameters.

These requirements are met in the current configuration that has three silicon pixel layers (Fig. 1.7) and ten silicon strip layers (Fig. 1.8) in the barrel. In the endcaps two pixel layers, three inner and nine outer forward discs made of silicon microstrip detectors are foreseen.

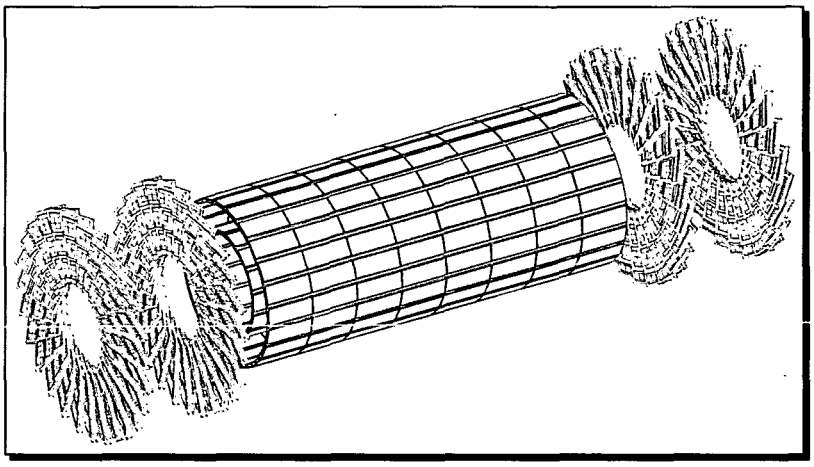


FIGURE 1.7: The pixel detector. This layout shows only two barrel layers, while the "final" version is conceived to include a third layer. The three layers are at an approximate radius of $r = 4/7/11$ cm from the beamline, respectively. The 2×2 pixel endcaps are at $z = \pm 35/47$ cm, respectively.

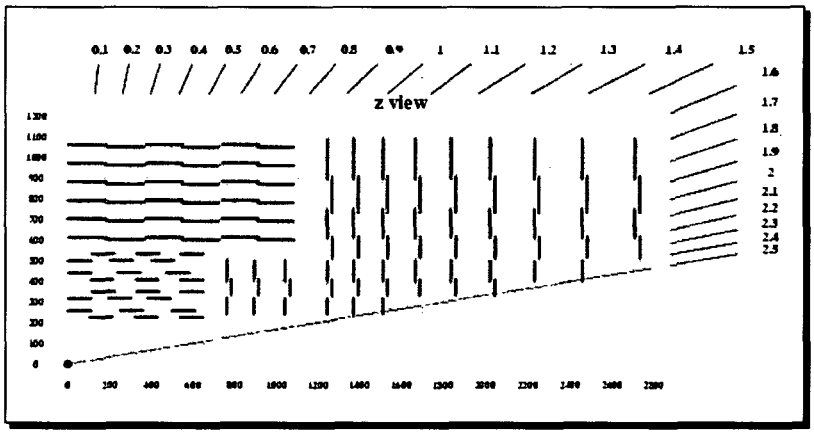


FIGURE 1.8: An $r - z$ view of a quadrant of the silicon microstrips of the inner tracker. Numbers are given in mm. Red lines symbolize single-sided modules; blue lines denote double-sided modules. Note the values for the pseudo-rapidity η around the plot; we can see clearly that the inner tracker mainly covers the range $|\eta| < 2.4$; the barrel detectors start at $|\eta| < 1.4$.

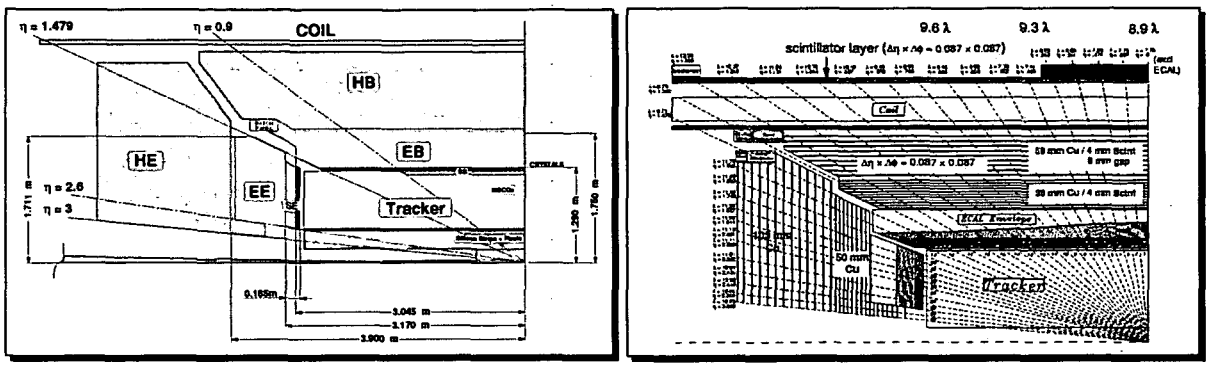


FIGURE 1.9: The electromagnetic and the hadronic calorimeters

1.2.2 The calorimetry

The CMS design foresees two calorimetry subsystems: an electromagnetic calorimeter (ECAL) and a hadronic calorimeter (HCAL). Calorimetry information is used for the definition of jet cones, which – in turn – are essential objects for many b-tagging strategies. Hence, calorimetry information must be taken into account for analysis of vertex reconstruction.

The electromagnetic calorimeter

The purpose of this subsystem is to measure the direction and energy of electrons, positrons and photons. This is achieved with over 80 000 lead-tungstate ($PbWO_4$) crystals. Lead-tungstate is a very responsive transparent scintillating radiation-hard material with a small Moliere radius and a short radiation length which allows for a very compact design. The ECAL itself consists of two different types: the barrel part, that covers a pseudo-rapidity range of roughly $|\eta| < 1.48$ and two endcaps (on either side of the barrel) that cover an η of up to 3.0. Electrons, positrons, and photons that enter the calorimeter produce an electromagnetic shower inside the crystals, which in turn triggers a scintillation light proportional to the energy of the original particles. The scintillation light is measured by vacuum phototriodes in the endcaps and by avalanche photodiodes in the barrel.

A preshower detector, consisting of lead radiators and two orthogonal planes of silicon strip detectors, is placed in front of the endcaps to help determine the multiplicity of an electromagnetic shower, i.e. the number of incident particles that produced a shower.

The hadronic calorimeter

The hadronic calorimeter (HCAL) is specialized in the measurement of direction and energy of hadrons and hadronic jets. The HCAL consists of four different types: the barrel (HB), the outer barrel (HO), the endcaps (HE) and the forward calorimeter (HF). HB and HE are within the radius of the coil, the HO is around it. HF is situated in the very forward

region of CMS, some 11 meters from the interaction region. HB and HE are *sampling calorimeters* made of 50 mm thick copper absorbers interleaved with 4 mm thick plastic scintillators that are read out with wavelength shifting fibers. Scintillation light is detected by hybrid photodiodes. HF is built of steel absorber plates.

1.2.3 The muon system and the magnetic yoke

A good reconstruction of muons is expected to be essential for many data analyses. It is the task of the muon system to identify muons and provide, together with the tracker, a precise measurement of their track parameters.

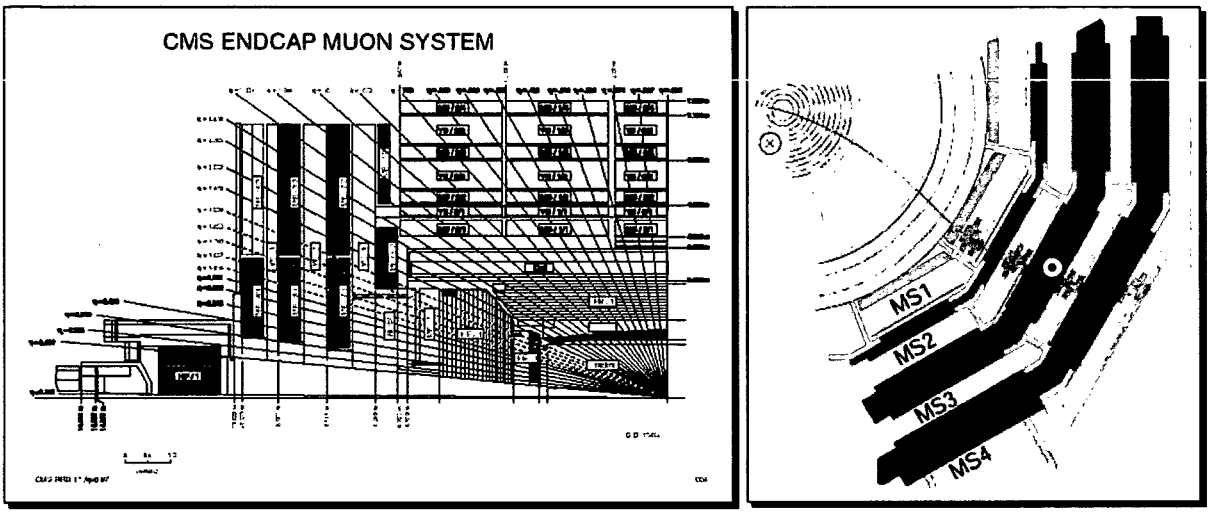


FIGURE 1.10: The muon system in CMS. Left: a quadrant of the barrel and endcap muon system. Right: transverse view of the barrel region. A muon that traverses the various muon stations is shown.

The magnetic yoke is one part of the muon system. It is built to capture the magnetic field for the "muon stations" that are embedded in the yoke. The muon stations contain the sensitive parts of the muon system. Different technologies are used in the different regions. Fig. 1.10 shows the schematic layout of the muon system.

1.3 Data flow

I cannot do't without comp[u]ters

William Shakespeare, "The winter's tale"

LHC is designed to produce collisions of proton-proton bunches at a frequency of 40 MHz. Every bunch contains 10^{11} protons. At the design luminosity of $10^{34}\text{cm}^{-2}\text{s}^{-1}$ each crossing produces on average 20 inelastic and 5 elastic collisions. One such *event* produces

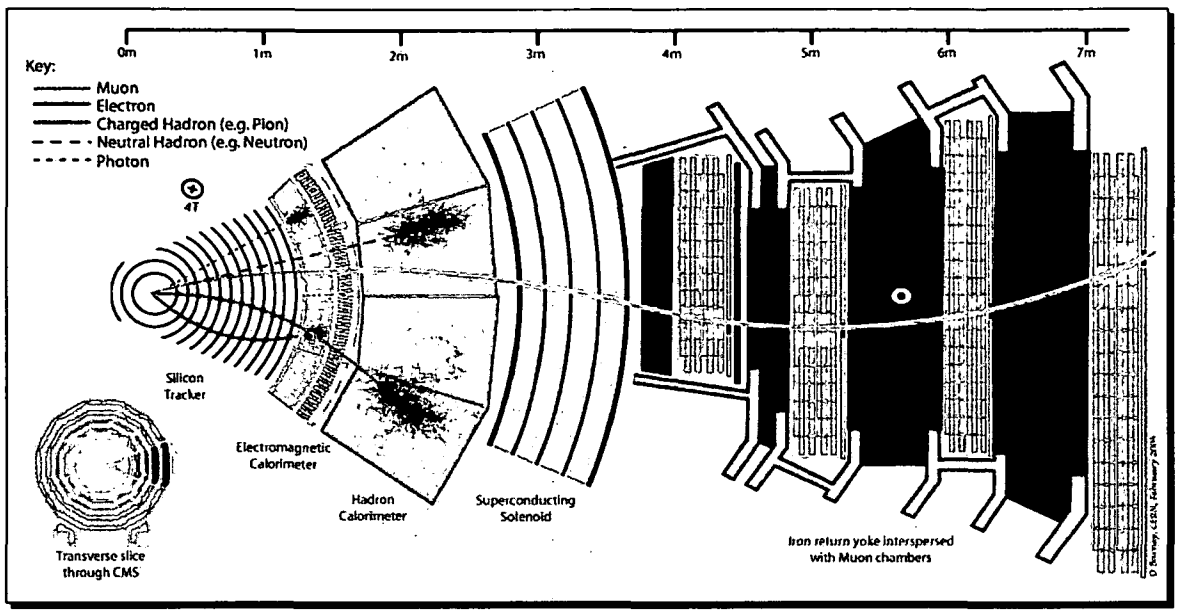


FIGURE 1.11: A transverse slice through CMS and typical paths of a few different particle types.

approximately 1 MB of raw data, which results in the equivalence of 40MHz × 1MB ≈ 40 terabytes per second. Surely no data storage technology can handle such data rates. Luckily a vast majority of the events is of no interest to physics. Interesting events can be filtered out. Event filtering in CMS is done in two levels: a level 1 (LV1) hardware trigger and a high-level software trigger (HLT) – see Fig. 1.12. Initial event data resides in front-end buffers and pipelines until the hardware trigger has reached a decision. If the event is to be kept, the data is transferred into various read-out buffers. In a next step an *event builder* combines an event's scattered data that are in the different read-out buffers and produces one single data structure per event. This consistent event data structure is then shipped to the filter farm. The farm will consist of some ≈ 5000 computers; the event rate at this stage is down to $\mathcal{O}(10^5)$Hz. Every event will be processed on a single CPU. Assuming that every computer has two CPUs we arrive at $10^5 s^{-1}/10^4 \approx 10 s^{-1}$ events per second that one CPU has to cope with, or 100 milliseconds per event. Based upon the final decision of the HLT, an event is either discarded or stored permanently. The output event rate of the HLT is estimated to be 10^2 Hz; this amounts to a daily 10^{13} bytes (10 TB) that need to be stored. Stored data is then analyzed offline in large world-spanning collaborations. The data itself will not be kept in a centralized place but spread around the participating institutes' storage resources. Grid technologies [5] will be used to manage the scattered data and the computing resources.

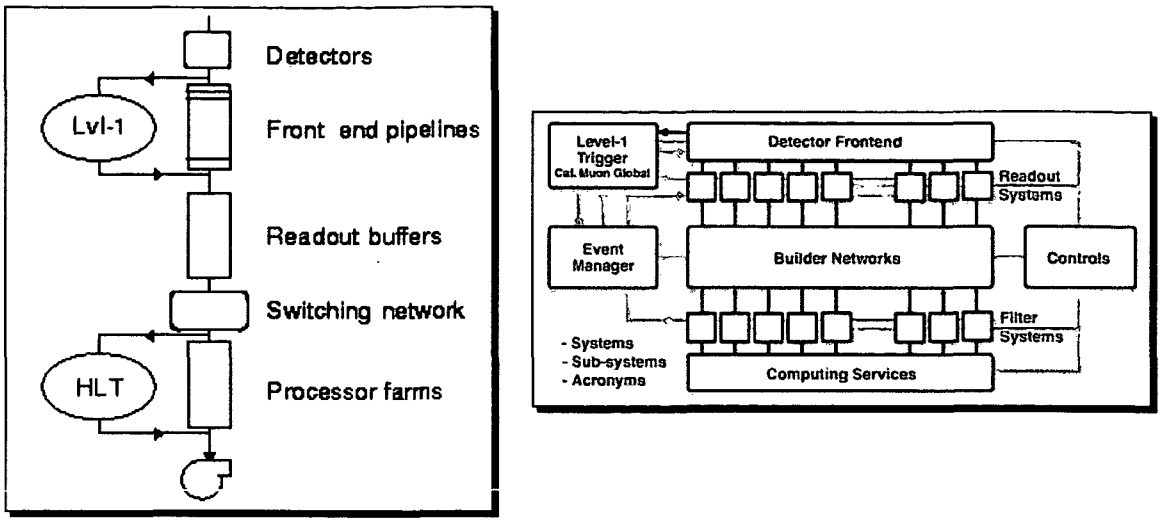


FIGURE 1.12: Trigger and data acquisition. Data flow (left) and design (right).

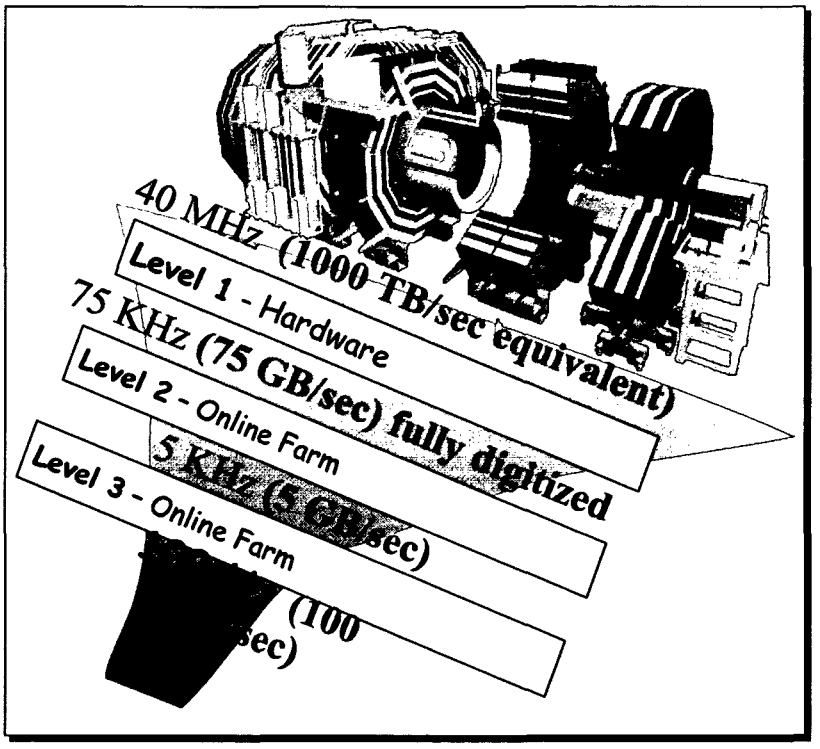


FIGURE 1.13: Data rates at the different trigger levels.

1.4 CMS offline software

Some of CMS offline software started in the old FORTRAN world. A few of the old projects that are now obsolete or being phased out (like CMSIM) give testimony of this legacy. The

more recent projects are all based on C++; the paradigm shift is finished. The current CMS software framework is heavily based on LCG software projects [9]. CMS software exploits LCG's grid software, its persistency solution POOL [14], as well as a large variety of libraries and tools that are available, e.g. via the SEAL project [16]. PAW [12] was replaced by ROOT [29] as the supported analysis (plotting and histogramming) software framework. Python language bindings have later been added to ROOT in version 4.

1.4.1 CMSIM

CMSIM [33] is a single large GEANT3 [59] based FORTRAN application for detector simulation (and reconstruction). Although OO successors exist already (see OSCAR, Sec. 1.4.6), it is still used for detector simulation in this thesis, mainly because its physics output can still be called more reliable. The community's migration from CMSIM to OSCAR took place only very recently.

1.4.2 PYTHIA

PYTHIA [88, 87] is an event generator that simulates standard model processes as well as a wide variety of non-standard physics, presented in Sec. 1.1. PYTHIA can be considered the default event generator in CMS; for this thesis no other event generator has been used.

1.4.3 COBRA

COBRA [66] (**C**oherent **O**bject-oriented **B**ase for **R**econstruction, **A**nalysis and simulation) forms the base of CMS specific software. Its central piece is CARF, the main application framework, see next subsection. Apart from the framework it also offers a few helper classes and interfaces to event generators, magnetic field, and data acquisition. For instance, the SimpleConfigurable class that will often appear in the thesis is part of a COBRA package. It allows one to have parameters in the code that are easily configurable at runtime via an "rc" file (.orcarc).

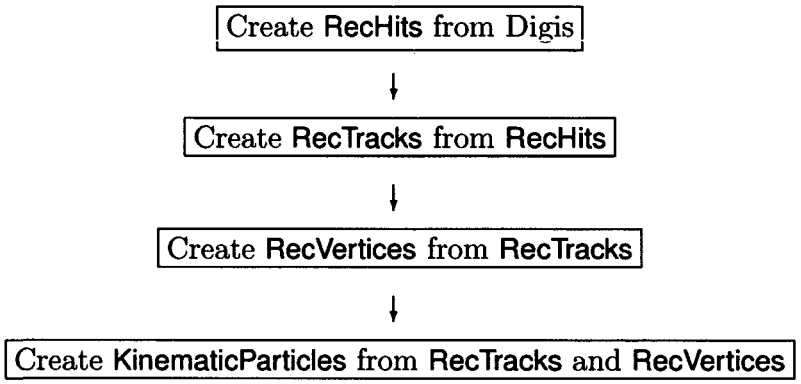
1.4.4 CARF

CARF [66, 67] (**C**MS **A**nalysis and **R**econstruction **F**ramework) is the main application framework for offline reconstruction as well as the high level trigger. It defines the main reconstruction event loop. To this end it utilizes and extends the *Observer* pattern [57]. CARF implements an "action on demand" principle; nothing is supposed to be done unless it is "needed". This, too, is implemented with the *Observer* pattern. This behavior will be emulated in the VertexFastSim package, see Sec. 2.2.2.

1.4.5 ORCA

ORCA [38, 91] (**O**bject oriented **R**econstruction for **C**MS **A**nalysis) surely is the flagship of the CMS software community. It is by far the largest artefact. Functionally it is based on CARF. ORCA is responsible for the complete reconstruction chain from the DAQ to physics objects. It has to supply both the reconstruction tools for the HLT and offline mode. Note that there will not be a single reconstruction strategy. Different tasks will employ different sets of algorithms. It is ORCA's task to provide a flexible environment that makes a wide variety of reconstruction algorithms easily accessible to the end user — the data analyst.

In the case of reconstruction in the tracker, a simple reconstruction chain looks like this:



In this figure the scope of the thesis would be the third box. (A description of the above classes will be given in Sec. 1.5.1).

1.4.6 OSCAR

OSCAR [37] (**O**bject oriented **S**imulation for **C**MS **A**nalysis and **R**econstruction) is CMSIM's modern object oriented successor, written in C++. The output of an event generator is the input for OSCAR, which traces the particles through the detector. OSCAR uses the GEANT4 toolkit [22, 60] for many of the computational steps, like tracking a charged particle through an external magnetic field.

1.4.7 FAMOS

FAMOS [24] stands for **Fa**st **Mo**nte Carlo **S**imulation. Very often, events need not be simulated at a granularity as fine as the "full" simulation packages like OSCAR or CMSIM. Full simulation is a CPU intensive task that is performed in several distinct steps. Various shortcuts to the simulation chain have been implemented in FAMOS. Fig. 1.14 shows several of those shortcuts.

1.4.8 IGUANA

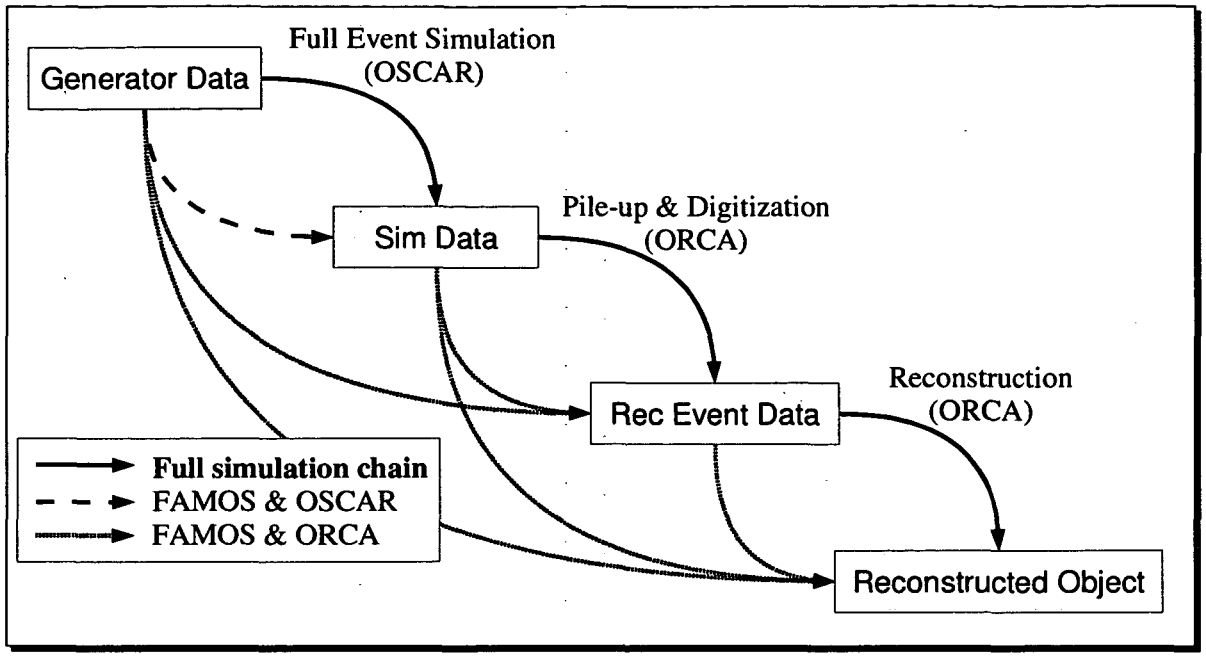


FIGURE 1.14: Several shortcuts that the FAMOS package offers compared to the full simulation chain. (Picture taken from [74]).

IGUANA [96] (**I**nteractive **G**raphics for **U**ser **An**alysis) is CMS's official visualization project. Despite it being an official CMS artefact, it aims to be a general, detector independent, extensible "non-intrusive" environment that provides a rich set of tools needed for visualization and analysis. It is, itself, based on standard libraries like Coin3D [4], Qt [15], and various LCG tools [9].

1.4.9 Track reconstruction in ORCA

Track reconstruction in CMS is described in great detail in [104]. The topics of track reconstruction and vertex reconstruction are closely coupled in CMS; the supervisor of this thesis contributed to many, if not all, major ideas for novel track reconstruction concepts that are implemented in ORCA. A Kalman filtering technique [50, 51] can be considered the classic algorithm. CMS with its noisy data has triggered the development and analysis of many novel algorithms:

- The Riemann fit [95, 93, 94, 55] is a viable alternative to the Kalman technique. It works by mapping the transversal coordinates of the RecHits on a Riemann sphere. The task of fitting a circle in the $R-\phi$-plane maps onto the task of fitting a plane intersecting the sphere. The latter is computationally much simpler than the original aim.

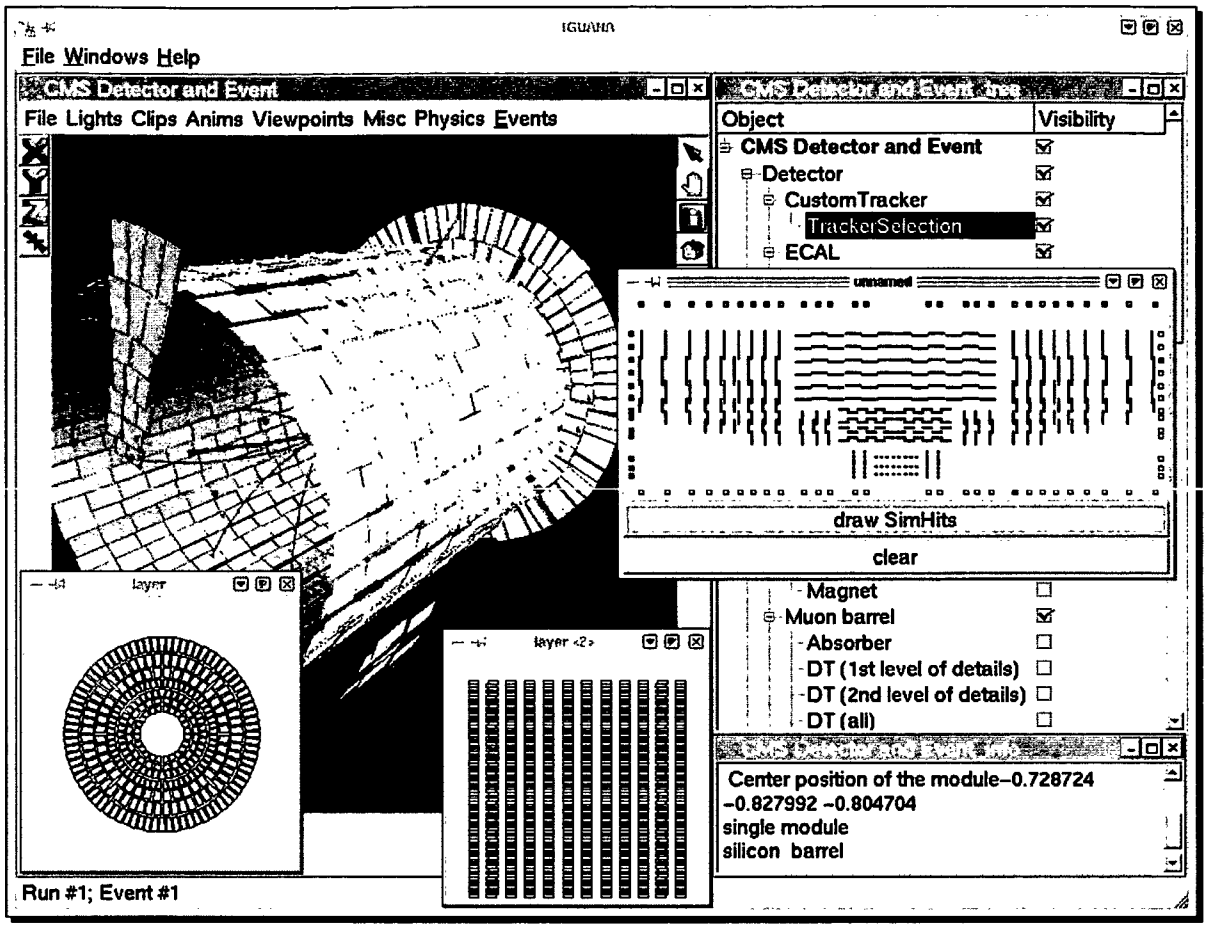


FIGURE 1.15: IGUANA screenshot, taken from the IGUANA home page. Parts of the tracker are visible, as well as a few tracks.

The final uses for the Riemann fit are still unclear. The Riemann fitter seems to make sense as an initialization for Kalman-filter based techniques. It is also interesting as a fast method for high energy tracks; the method seems to have its problems in the presence of multiple scattering.

- Robustifications of the classical Kalman filter [54, 75, 92] are interesting in the presence of noise and imperfect data, as is the case at LHC. The deterministic annealing filter (DAF) [104] seems to be a very powerful general purpose algorithm. It works by assigning track-association weights to the RecHits. Hits that are incompatible with a track candidate are down-weighted. The adaptive vertex fitter and the multi vertex fitter that will be presented in Sec. 5.4.2 and 5.5 are closely related to the deterministic annealing track fitter.

- The Gaussian sum filter (GSF [21, 20] is able to deal with mixtures of Gaussian dis-

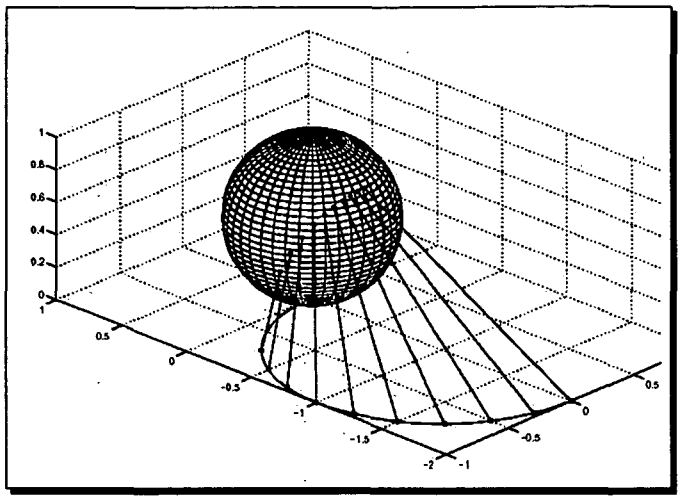


FIGURE 1.16: The Riemann fit maps the hits onto a Riemann sphere where a plane is fitted into the data set.

tributions. It is usually implemented as Kalman filters that run in parallel, one filter for every component in a Gaussian mixture. Assignment probabilities are assigned to the Gaussian components according to a specific metric. Often the combinatorics needs to be actively kept low ("trimming") because the number of Kalman filter rises exponentially.

The most promising use case is the reconstruction of electron tracks in the presence of bremsstrahlung photons.

The GsfVertexFitter [53] that will be briefly mentioned in Sec. 5.7, is the Vertex system's counterpart to the GSF Track Fitter.

1.4.10 Online versus offline reconstruction

The constraints for online reconstruction algorithms differ from those for the offline algorithms. CPU time plays a major role for the first case, while it is only moderately important for the second case. So clearly one will employ different algorithms for the two different use cases. Still it is desirable to have a framework that allows online and offline algorithms to be interchangeable. ORCA complies with that design. All algorithms that are introduced in this thesis can in principle be used for online and for offline use. The time budget dedicated to online vertex reconstruction is estimated to be 40 ms per event in 2007. Applying the inverse of Moore's law, one arrives at a 300 ms on a 1 GHz PC. See also [39] (p.9) for a few general comments on event filtering and time budgets in CMS.

1.5 The vertex subsystem

Of all of ORCA's subsystems, the vertex subsystem [11] plays the most important role for this thesis: almost all of this work's contribution to CMS is in the form of code that resides there. This section shall give a summary of the most important classes and concepts.

1.5.1 Data objects

Data objects are instantiations of classes that represent a specific consistent set of data. Such well-defined, self-contained data objects implement the OO paradigm of *encapsulation*. Here is a list of the most frequently used data objects in the vertex package:

- **RecHit** A **RecHit** represents the information measured by a detector. It is fairly general, for instance the number of dimensions is not fixed. It is also a *composite* object — a single **RecHit** can consist of several **RecHits**. In the tracker, a **RecHit** is used to represent detector signals in the pixels and the microstrips. **RecHits** are never accessed directly in the vertex package. Their relevance in this thesis is confined to the usage of one specific **TrackAsssociator** – see Sec. B.5.1.

- **GlobalPoint** represents a simple point in "global" Euclidean 3d space.

- **RecTrack** A **RecTrack** represents a reconstructed particle track. It has the information of the trajectory state vector. This vector can be propagated to any point on the track. It also stores the **RecHit** collection that is part of the track. It is a variant of a *composite* object: it contains a set of trajectory state vectors; if queried for *the* trajectory state vector, it returns a collapsed vector. This feature is used in the Gaussian Sum Track Filter (GSF). The interface of the **RecTrack** class is heavy; a lot of infrastructure, like propagating the track to various geometric objects (points, lines, surfaces) is part of the **RecTrack** interface. It is also a *proxy* object - the **RecTrack** class itself does not contain the data. It can be copied without overhead; value semantics should be used.

- **TrajectoryStateOnSurface** describes a state vector that is "bound" to a certain surface. It is reference counted; it should be used by value. It is a *composite* object. A **TrajectoryStateOnSurface** can be asked for its components. In case it does not have any, it returns itself as the only component.

- *TkSimTrack* An object only available in Monte Carlo simulations. It represents the track of a simulated particle that is sufficiently long-lived to be observable. A *TkSimTrack* contains the information of particle momentum, production and decay vertices, charge, and generator particle (i.e. particle type and the like). The abstract *TkSimTrack*, too, includes a few methods for propagating the track.

- *TkSimVertex* is also a Monte Carlo object. It represents the decay vertex of certain particles. *TkSimVertices* are created only if the parent particle is long-lived. The

class contains pointers to the mother particle and the decay products, as well as the vertex position.

- *TkSimEvent* represents the collection of the *TkSimTracks* and *TkSimVertices*.
- *TkRecEvent* contains (currently) nothing but the collection of RecTracks.
- *G3EventProxy* is the proxy class of a GEANT3 event. This is the object that the CARF framework dispatches to all EventAnalysers; all event information is to be retrieved from it. Specifically it can be asked for the TkSimEvent and the TkRecEvent objects.
- RecObj is a generic reconstructed object. It is implemented in CARF. Reconstructed objects that are supposed to be streamed onto DSTs (Data Summary Tapes) need to derive from this class.

This list shows the most important data classes that are defined in the vertex package:

- RecVertex A RecVertex represents a reconstructed vertex. It contains the location and the error of the interaction vertex, plus the tracks that were used to fit the vertex. The knowledge of the vertex position can be used to refit ("smooth") the track momenta. A RecVertex keeps both the original and the refitted tracks.
- CachingVertex A CachingVertex is similar to a RecVertex object, except that it keeps some additional (redundant) information for CPU performance reasons. It can keep e.g. the weight matrix — the inverse of the covariance matrix.
- *LinearizedTrackState* Given a RecTrack and a GlobalPoint, the track is linearized around the point and the information is kept in this class. The specific coordinate system in which the track is linearized depends on the implementation class. This class, too, is a *composite* class.

1.5.2 Algorithm classes

Algorithm objects are not defined by what they *contain* — very often, in fact, algorithmic objects don't include any data members. They are defined by what they *do*. In order to enjoy maximum freedom in the choice of algorithms employed by the user, all important algorithm classes have an abstract interface — they derive from an abstract base class. These abstract interfaces implement *polymorphism*: no matter what e.g. a specific VertexFitter does internally — we know we can use it to fit tracks into vertices.

The list below names the most important algorithmic concepts that are referred to in the thesis but are not part of the vertex package:

- *TrackFinder*: A track finder solves the pattern recognition problem of finding (and fitting) the RecTracks in a certain region of the detector.

- *TrackReconstructor*: A track reconstructor is the "glue code" that couples a *TrackFinder* to the CARF framework.
- *EventAnalyser*: The CARF framework dispatches the events (the G3EventProxy objects) to every *EventAnalyser* that has registered itself to the framework. Every "regular" user code needs to have such an *EventAnalyser* to couple the analysis code to the framework.
- *TrackAssociator*: A TrackAssociator is an object that receives a list of TkSimTracks and a list of RecTracks upon construction. When queried, it can then return lists of TkSimTracks that are most compatible with a given RecTrack, and vice versa.

Now follows an enumeration of the most important algorithm concepts that are defined in the vertex package:

- *VertexFitter*: A VertexFitter is an algorithm that fits a RecVertex from a given set of tracks: vector<RecTrack> → RecVertex
- A *VertexReconstructor* finds and fits RecVertices from a given set of tracks: vector<RecTrack> → vector<RecVertex>
- *VertexUpdator*: All VertexFitters that add tracks sequentially (like any Kalman filter technique) employ a VertexUpdator that updates a vertex candidate with one track.
- *LinearizationPointFinder*: Almost all VertexFitters need an initial rough guess of the vertex location. A LinearizationPointFinder is an algorithm that delivers just that: vector<RecTrack> → GlobalPoint
- A *VertexTrackCompatibilityEstimator* computes that compatibility between a track and a vertex.
- A *ClosestApproachOfHelices* algorithm computes the points of closest approach between two tracks.
- *VertexAssociator*: A VertexAssociator is the vertex package's counterpart to a TrackAssociator. For a TkSimVertex it returns a list of compatible RecVertices and vice versa.

1.6 Scope of this thesis

It is the aim of this thesis to develop, implement and analyze novel algorithms for the estimation of interaction vertices from tracks (known as "vertex fitting") and for the problem of grouping the tracks into sets that share points of origin (a.k.a. "vertex finding"). Contrary to a typical algorithmic thesis it was necessary to not only implement an algorithmic prototype that verifies the algorithmic properties predicted by theory. Instead, code that

works in an online and an offline situation is demanded. Some of the algorithms presented in this thesis might easily run of the order of 100000 times a second once CMS has started to take data; thus reconstruction code needs to be very robust. It is safe to say that a few of the implementations in this thesis will find their way into widespread use in the CMS community and hopefully beyond; they will have to run (often unsupervised) over a huge amount of data. These challenging technical aspects should also be taken into account when judging the "value" of this thesis.

In yet greater detail, the scope of this thesis comprises:

- Design, implementation and verification of mode finders (Ch. 3). The need for mode finders emerges in more than one place in this thesis. An entire chapter is devoted to identifying a few strategies; the implementation is also presented.
- Design, implementation, and performance analysis of a few linearization point finders. Vertex fitters need linearization point finders for an initial rough guess as to where the vertex is to be found. A few ideas will be presented; they will then be compared in a few relevant physics cases.
- Implementation, and verification of novel vertex fitting algorithms (Ch. 5). The noisy LHC environment makes the study of robust statistical algorithms necessary. A few robustification strategies were identified and implemented. The implementation was verified with a few simple statistical tests.
- Development of a fast simulation package tailor-made for vertexing needs. The package comprises very fast "unphysical" event generators (VertexGuns), fake TrackReconstructors that are fully controlled by the user (RecTrackSmearer), and a framework that renders the event source almost invisible for user code (VertexFastSim).
- Design and development of vertex finding algorithms (Ch. 4). A few pattern recognition algorithms known in literature have been adapted to the problem of vertex finding. They were then implemented and tested.
- Performance analysis of vertex fitters and vertex finders in relevant, realistic physics cases (Ch. 6). Vertex reconstruction strategies were compared, both in "geometric" terms and in terms of b-tagging efficiencies. Resolutions, pulls, and failure rates of vertex fitters were compared.
- Design, development, and verification of an algorithm that computes the points of closest approach of two tracks (Sec. A.1).
- Development of a framework that automatically tunes one parameter of a vertex finding algorithm against an objective function that is configurable by the user (Sec. B.1).
- Development of a tool that allows to dump variables into a file in a very simple manner. The harvesting concept (Sec. B.2) is supposed to make debugging easier.
- Development of a simple visualization tool (Sec. B.3).

1.7 Writing conventions

RecTrack denotes the implementation class called "RecTrack", *VertexFitter* is an abstract class (a.k.a. an "interface"). `MaxEvents=1` denotes a configurable quantity (i.e. a line in an .orcarc file). Design patterns, such as the *singleton* [57] pattern are denoted *in italics*. C++ source code is written as follows:

```
cout << "Hello World" << endl;
```

CHAPTER 2

Simulation

Science may be described as the art of systematic over-simplification.

K. Popper

Collider experiments rely heavily on realistic simulation of event data. Ultimately most data analyses end up comparing real data with simulated data with a known and well-defined underlying physics model. For the development and analysis of reconstruction algorithms it is often necessary to produce data with controlled event kinematics and/or track reconstruction. To this end the VertexFastSim package has been developed. It allows to control every aspect of an event's RecTracks - both the kinematics and the pseudo-reconstruction (the "track smearing") are entirely put into the user's hands.

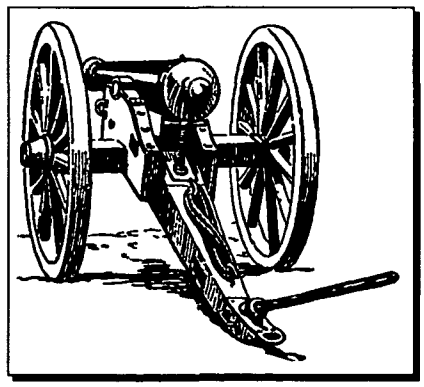


FIGURE 2.1: A cannon, the historical prototype of the modern VertexGuns. This military technology is at last brought to civil use, just like in the case of the Kalman filter.

2.1 Full simulation

In order to perform realistic performance analyses of reconstruction algorithms, one needs realistic data. Standard "full sim" events in CMS are often generated via PYTHIA and CMSIM. More recently OSCAR has taken over as CMSIM's object oriented successor. For fast simulation, FAMOS is gaining popularity in the CMS community. Certainly, for physics analysis many event generators are used; this being an algorithm oriented thesis, it can safely be assumed that the results presented in this thesis do not depend much on the choice of event generator.

A certain number of channels were chosen as test channels for various algorithmic analyses; lacking official data sets, some had to be produced specifically for this thesis by the author. In the time period of writing the thesis, CARF's persistency layer had periodically gone through tremendous non-compatible changes; very often all the generated data that had taken CPU months to produce had to be discarded and reproduced.

The next section shall list the data (both produced for the thesis and "official" productions) that had been used versus the end of the thesis. The data described here is usable from ORCA_8_2_0; they are in a world readable afs directory. The reader is invited to use the data for his own purposes.

If not noted otherwise, the event samples used throughout this thesis are from these productions.

Note that past experience suggests that the information content of this chapter is highly volatile.

2.1.1 Data produced for the thesis

A few event topologies were produced specifically for this thesis. The data were generated using PYTHIA and CMSIM. The "digis" were produced in the standard ORCA procedure, the tracks were reconstructed with the default track reconstructor (combinatorial Kalman filter). The channels produced are:

- $b\bar{b}$, $|\eta| < 1.4$, 50 GeV – Production of a b and a $\bar{b}$ quark; quark confinement demands that the quarks hadronize almost instantaneously. Two B mesons are produced that travel on average 2mm (lab frame) before they decay, producing two distinct and collimated jets with total jet momenta equal to the momenta of the original b-quarks. This makes it a good benchmark test for secondary vertex finding algorithms. In order to create a more realistic use case, the events are filtered according to calorimetry information; tracks outside of jet cones are discarded. This sample covers the barrel region of the tracker ($|\eta| < 1.4$).

- $b\bar{b}$, $|\eta| < 1.4$, 100 GeV – The same event topology is produced with different jet energies and pseudorapidities. This enables us to study of the dependency of vertex reconstruction as a function of the two aforementioned parameters.

- $b\bar{b}$, $|\eta| < 1.4$, 200 GeV

- $b\bar{b}$, $1.4 < |\eta| < 2.4$, 50 GeV – Same as the above, only for the high η region. A look at Fig. 1.8 reveals that the interval for η corresponds with the forward region of the CMS silicon tracker.
- $b\bar{b}$, $1.4 < |\eta| < 2.4$, 100 GeV
- $b\bar{b}$, $1.4 < |\eta| < 2.4$, 200 GeV
- $c\bar{c}$, $|\eta| < 1.4$, 100GeV – Production of a charm quark and its anti-quark, analogous to the $b\bar{b}$ case. Charmed mesons have a much shorter lifetime; they travel only ≈ 0.5 mm in the detector. They, too, produce two jets, but very close to one another and to the primary vertex. This will be used as a very tough "scenario" for vertex finding algorithms, and as a background to $b\bar{b}$ events.
- $q\bar{q}$, $|\eta| < 1.4$, 100 GeV – Production of a light (u, d or s) quark and its anti-quark. No search for secondary vertices is performed. This event topology thus serves as a primary vertex contaminated with tracks from decays of strange mesons and baryons.
- $J/\psi\ \phi$ – At ORCA level we shall filter for $J/\psi\ \phi \rightarrow K^+K^-\mu^+\mu^-$. The topology is used as a highly collimated four-prong decay vertex. This serves as a use case for vertex fitting algorithms.
- $h^0 \rightarrow \tau^+\tau^- \rightarrow \pi^+\pi^-\pi^+\pi^-\pi^+\pi^-$ – Light MSSM neutral Higgs, decaying into two τs, both of which decay into three-prong $\pi^\pm\pi^+\pi^-$ topologies. At analysis level usually one of the three-prong vertices is filtered out; the other one is used for testing fitting algorithms.

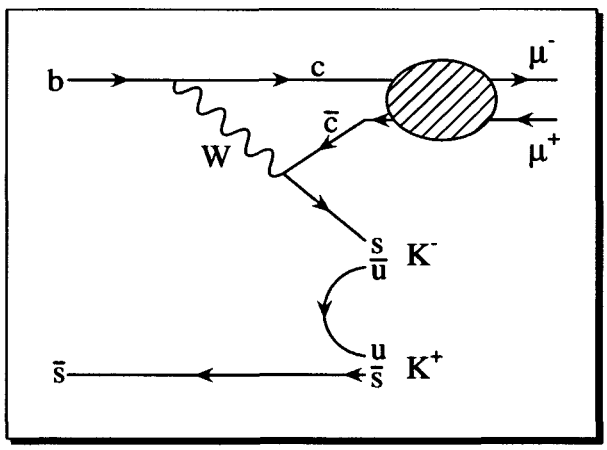


FIGURE 2.2: Diagram of the decay $B_s \rightarrow J/\psi\ \phi \rightarrow \mu^+\mu^-K^+K^-$. This is topology will appear frequently in this thesis, e.g. in the fitter tests.

These data can be found at `/afs/hephy.at/project/cms/simul/820`. Please contact the author for a more precise description.

2.1.2 Other data sources

The official test samples

Every recent ORCA release always come with a few test samples. These samples, albeit being small, are very useful for small analyses and code verification. In the last few years, they have been the most reliable source of event data. Information on the test samples is available at `http://cmsdoc.cern.ch/orca/testdata.html`.

The b-tagging samples

The b-tagging group has also produced its own data, specific to b-tagging purposes. The events generated include three channels:

- a "bg" channel - in which a b-jet "recoils" from a gluon jet.
- a "$c\bar{c}$" channel, and
- a light quark channel.

The former is used as a signal for b-tagging purposes; the latter two serve as background. For any detailed information contact the BReco package administrator.

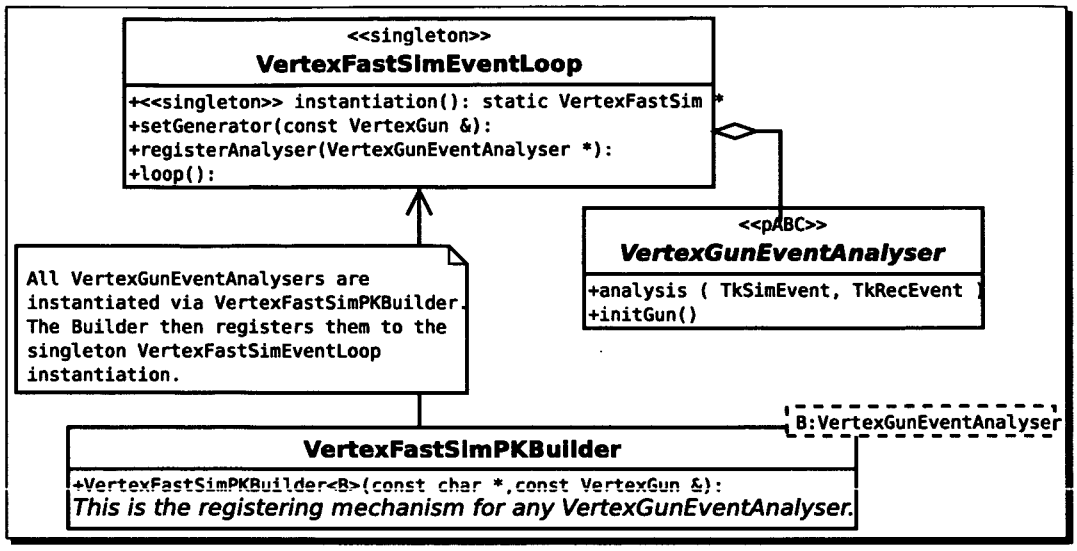


FIGURE 2.3: The VertexFastSim framework.

2.2 Fast simulation

2.2.1 Motivation

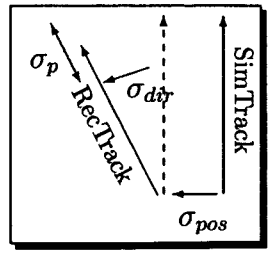
Realism of simulated data is not always a top issue. For verification of the implementation of vertex reconstruction methods one wants to have complete control over the reconstructed tracks. The VertexFastSim package addresses such use cases; it can be seen as a very fast event generator that completely ignores physics realism. Within the framework that is presented in this section, it is the user who decides where decay vertices appear and with what track multiplicity. The user also takes control over the track momenta and the way tracks are "reconstructed" — reconstruction is simply done by "smearing" the SimTracks according to a specific statistical model.

2.2.2 The framework — VertexFastSim

2.2.3 Event generation

Track smearing

When creating RecTracks from SimTracks the tracks need to be "smeared" according to a well-defined statistical model. Smearing is implemented as a three-staged process:



- Translation: The local (2d) position of the TrajectoryStateOnSurface is shifted according to $N(0, \sigma_{\text{pos}})$.

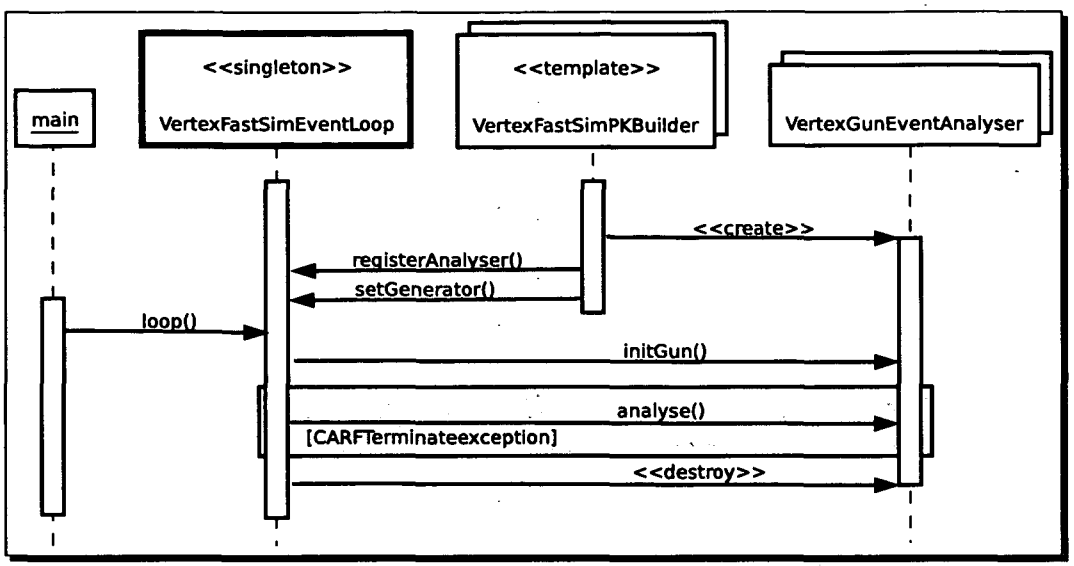


FIGURE 2.4: The VertexFastSim sequence.

- Rotation: The (2d) direction of the state vector is changed according to $N(0, \sigma_{\text{dir}})$
- Stretching: The vector is resized according to $N(0, \sigma_{\text{p}})$.

This procedure is done in the class **GaussianRecTrackSmearer**. For many applications, though, one wants non-Gaussian or mis-measured track errors. To achieve this, the abstract base class ***RecTrackSmearer*** was introduced, along with a **TwoGaussiansRecTrackSmearer** a *composite* class that combines two ***RecTrackSmearers*** into another ***RecTrackSmearer***. With this simple design **any** track error distribution can in principle be approximated, since any error can be modeled as a mixture of n Gaussians. Practically a mixture of more than two Gaussians has never been needed. Thus, it is not only *in principle* a sufficient design, but also *in practice*.

Introducing correlations

For a few sophisticated use cases well-defined correlations in the track parameters were required. To this end the **GaussianRecTrackSmearer** was extended; now the user can optionally supply a 5-by-5 correlation matrix.

Hiding differences

In order to further hide the differences between the **VertexFastSim** framework and CARF's (much more powerful) functional equivalent, a class **MultiObserver** was introduced. It inherits both from **VertexGunEventAnalyser** and CARF's ***EventAnalyser***. That way, the same

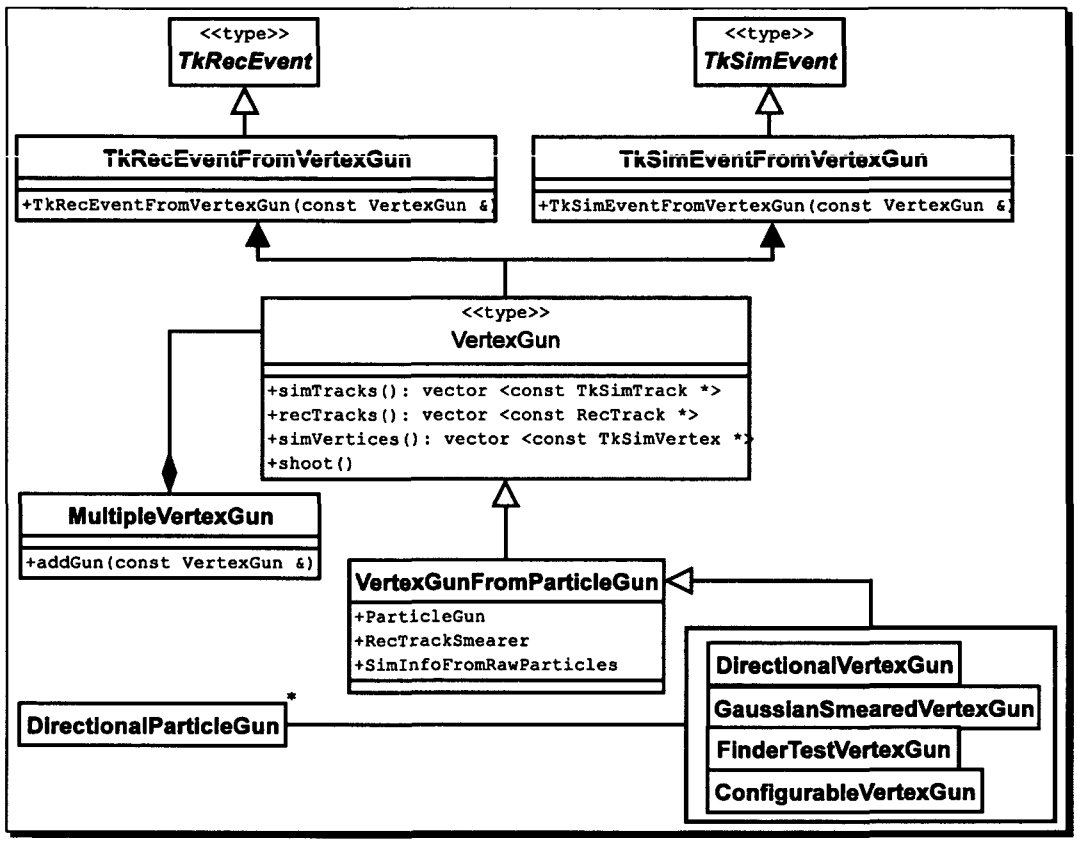


FIGURE 2.5: UML class diagram of the VertexGun facilities

analysis code can be used in both frameworks; which one is used is only a matter of which package builder is employed — PKBuilder or VertexFastSimPKBuilder. The usage of the builders is identical. Making this transition between frameworks a runtime option instead of a compile time option is a desirable future development.

Track associators

TkSimTracks and RecTracks are usually associated using their track parameters, or the list of their hits. In the case of the VertexFastSim framework there is an additional option: the "source" TkSimTrack from which a RecTrack is created is defined, unique, and known! This enables one to write a perfect TrackAssociator. This code exists – it is part of the VertexFastSim package, and the TrackAssociator is called TrackAssociationByMap. The e.g. ConfigurableTrackAssociator knows about it, it can be switched on at runtime via `ConfigurableTrackAssociator:Associate= ByFastSim`. A proper usage of the VertexFastSim package enables these associators by default (i.e. the default values of `ConfigurableTrackAssociator:Associate` and `VertexAssociationToolsFactory:-TrackAssociator` are automatically changed to the value `ByFastSim`)

Testing track smearing

In order to verify the correct statistical behavior of the track smearing code, the standardized residuals of the track parameters were measured. And indeed, the sigmas of the standardized residuals were all close to 1.0.

2.2.4 Standard scenarios

In order to have a fast means to test vertex finding algorithms against simple but useful events, we decided to standardize specific VertexGun configurations and distribute those frozen configurations as a small set of classes. The class FinderTestVertexGun implements this concept, ConfigurableFinderTestVertexGun makes the specific choice of the scenario a runtime option. Currently the following standard scenarios are defined: "easy", "realistic", and "tough":

"Easy" scenario

This is a very simple scenario, with two cleanly separable vertices; the jets of both vertices are perpendicular to the distance vector between the vertices. This scenario only serves debugging purposes.

	Primary vertex	Secondary vertex
Number of tracks	9–11	4–5
Total momentum	30 GeV	20 GeV
Track smearing, position	50 μm, Gaussian	50 μm, Gaussian
Track smearing, momentum	3 mrad, Gaussian	3 mrad, Gaussian
Angular spread	15°	15°
Jet direction	Perpendicular	Perpendicular
Distance between vertices	1 mm – 1.0 cm	

The jet direction is given with respect to the vector from the primary vertex to the secondary vertex.

"Realistic" scenario

This scenario has a primary vertex with tracks emerging from it at all angles. The secondary vertex is very displaced from the primary vertex and has an attached jet parallel to the line connecting the two vertices.

	Primary vertex	Secondary vertex
Number of tracks	9–11	4–5
Total momentum	30 GeV	20 GeV
Track smearing, position	50 μm, Gaussian	50 μm, Gaussian
Track smearing, momentum	3 mrad, Gaussian	3 mrad, Gaussian
Angular spread	180°	15°
Jet direction		Parallel
Distance between vertices	1 mm – 1.0 cm	

"Tough" scenario

This scenario is roughly modeled after a $b\bar{b}$ or a $c\bar{c}$ event. From the primary vertex a B meson emerges that decays some $100\mu\text{m} - 1mm$ "later" into another particle jet of 4–5 particles. Fig. 2.6 visualizes this topology; we can see two vertices, a red primary vertex, and a yellow secondary vertex. The tracks associated with the red vertex a drawn in cyan. Note in comparison the two innermost detector layers drawn schematically; these layers are where the first measurements are obtained. The innermost layer has a radius of ≈ 2.5 cm.

FIGURE 2.6: The "tough" scenario. One can see the primary vertex (red dot), the secondary vertex (yellow dot), the primary tracks(blue lines), the secondary tracks(cyan lines), and the two innermost detector layers (gray circles).

	Primary vertex	Secondary vertex
Number of tracks	9–11	4–5
Total momentum	30 GeV	20 GeV
Track smearing, position	50 μm, Gaussian	50 μm, Gaussian
Track smearing, momentum	3 mrad, Gaussian	3 mrad, Gaussian
Angular spread	15°	15°
Jet direction	Parallel	Parallel
Distance between vertices	100 μm – 1 mm	

2.2.5 VertexGunFromFile

One VertexGun was implemented that reads its data from a file. It is called the VertexGunFromFile; for reading the data it uses a DataSeeder (see subsection B.2.9), writing is done via the object harvesters (subsection B.2.8). The DataHarvesters currently cannot save on a per-event basis. Thus, the current implementation of the VertexGunFromFile relies on the assumption that the data are stored into the file in the right sequence. This assumption may not generally hold true. This is why currently this approach only works if the data are stored in text files, not in ROOT files. A future development (see subsection B.2.11) might address this issue in a truly general and efficient way.

So what is the use of such a gun, if, indeed, for realistic data one would not use this facility but access the Monte Carlo data via CARF? The VertexGunFromFile has three important advantages. It is much faster - since it concentrates on a much more specific task, it can read ASCII files (editable with any text editor!), and it is not subject to changes in CARF's persistency layer.

2.2.6 Verification, comparison

The *VertexGun* implementations were (partly) verified with the *VertexFitters*, and vice versa. Since all *VertexFitter* exhibit the theoretical properties that were predicted on a theoretical basis, we conclude that either the Fitters and the Guns work appropriately, or they have errors that cancel each other. The statistical properties that were explicitly checked were:

- The χ^2-distribution of single, clean vertex events, fitted with a linear method.
- The χ^2-probability of single, clean vertex events, fitted with a linear method.
- The standardized residuals of single, clean vertex events, fitted with a linear method.
- The χ^2-distribution, the χ^2-probability, and the standardized residuals of single vertex events with a well-defined number of outliers; fitted with both linear and non-linear methods.

"Clean" in this context means that the error of the track parameters is Gaussian, and that the reported errors and the "real" errors (i.e. the ones used for smearing) are identical.

A performance test was done with ORCA version 7_1_3; the generation of 5000 VertexGuns of the "tough" scenario takes 8 seconds; this amounts to 1.6 milliseconds (real-time) per event on a 1 GHz SMP computer. This compares favorably with the "regular" way of obtaining MC data: a setup that measured the time of accessing 50 $b\bar{b}$ events on a local hard disc, plus track reconstruction, took 370 (real-time) seconds on the same machine, which results in 7 seconds per event.

The *VertexGuns* thus produced similar data about 5000 times faster in this specific setup. Surely, for serious analyses one should never use the VertexFastSim framework; the resulting RecTracks are far too unrealistic. But in many cases *VertexGun* data are fully sufficient; in fact sometimes one even wants full control over the statistical properties of the input. This can *only* be achieved via the *VertexGuns*.

CHAPTER 3

Mode finding

The mode of a set of data is the value in the set that occurs most often. In the presence of continuous data it is the maximum in the "parent" distribution underlying the data. Mode finding is mathematically not concisely defined. In this thesis it will refer to the task of estimating the maximum of a data set's (non-defined) distribution, without making any specific assumptions about the distribution itself.

This chapter is divided into five sections: the first section motivates this chapter. The next two sections treat the special case of having one-dimensional data. Finally, the last two sections discuss algorithms that operate in three dimensions.

3.1 Motivation

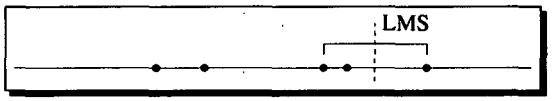
Mode finding will be used in a few different contexts in this thesis. The chapter on vertex finding (Ch. 4) will introduce points that represent **RecTracks** — the apex points. Finding these points represents a pattern recognition problem in one dimension. The **Clusterizer1D** that will be presented in this chapter, is used for apex point finding. The vertex fitting task, on the other hand, is confronted with the problem of finding a good initial vertex seed. One class of such **LinearizationPointFinders** is based on the points of closest approach (PtCAs) between track pairs. These PtCAs serve as the input for a mode finder in three dimensions. The output of this algorithm is a first guess of the vertex position. The one-dimensional mode finding package is written in a templated fashion. It is also used for finding the primary vertex along the beamline [41].

3.2 Mode finding in one dimension

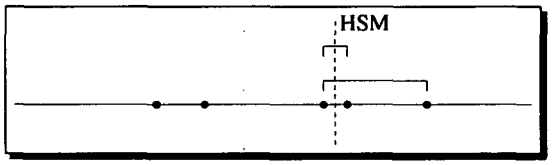
- **LMS** The "Least Median of Squares" estimator [84]. The LMS algorithm *per definition* minimizes the median of squared residuals.

$$\hat{\beta}_{LMS} = \underset{\beta}{\operatorname{argmin}} \operatorname{med} r_i^2(\beta) \tag{3.1}$$

No general algorithmic solution to this problem is known, except for a fully combinatorial approach. One suggestion for a faster algorithm has been to randomly select but a few combinations of data points [84]. For the special case of one-dimensional data literature knows of a simple and fast solution: in this case the LMS estimator is the midpoint of the smallest interval that covers at least 50% of all data points. Algorithmic complexity, assuming sorted input: $\mathcal{O}(n)$

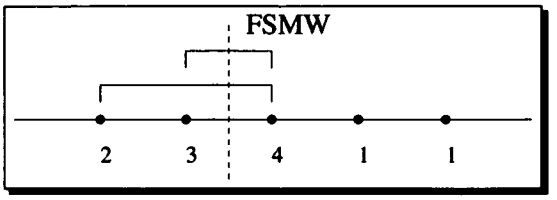


- **HSM** The "Half Sample Mode" [26] can be formulated as a recursive LMS estimator: we iteratively perform an LMS on the points in the interval of the last interval supplied by the LMS estimator. Algorithmic complexity, assuming sorted input: $\mathcal{O}(n \log_2 n)$

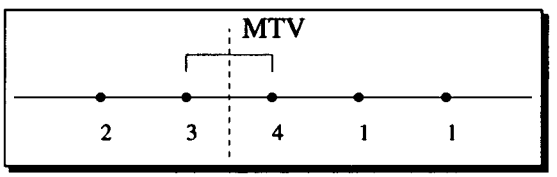


- **FSMW** The "Fraction-of Sample Mode with Weights"-estimator is a simple generalization of the HSM: instead of looking for an interval that covers $\geq 50\%$ of the data points we generalize to a search for the interval with k percent of the data points. Additionally we associate "weights" with the data points. Instead of searching for the smallest interval we now search for the smallest "weighted" interval, where a weighted interval is defined as the length of the interval divided by the sum of all weights of contained data points. The implementation allows - for performance reasons - to ignore the weights until the "sub-sample size" is below a certain threshold.

Complexity, assuming sorted data: $\begin{array}{ll} \mathcal{O}(n^2 \log_{(1/f)} n) & \text{(below user-supplied threshold)} \\ \mathcal{O}(n \log_{(1/f)} n) & \text{(above threshold)} \end{array}$



- **Kernel estimator** A kernel estimator [85] is a generalization of an average shifted histogram. It introduces a "kernel" function — a pdf with which the data points are "smeared out", then superimposed.
- **Deterministic annealing** The deterministic annealing algorithm [82, 83] introduces the metaphor of a thermodynamic system into the 1d mode finder problem. The thermodynamic system that consists of the data points "cools down", prototypes move and split with each "phase transition". The concept has been tried in the vertex finding problem, Sec. 4.5.5.
- **Gravitational clustering** [72] Every data point exerts a force of attraction on every other data point. The force of attraction follows a simple Newtonian $\frac{1}{r^2}$ law. Data points are considered impenetrable — a clash of two points results in a larger data point with the total weight being the sum of the weights of its constituents.
- **MTV** The "maximal two values" algorithm is a very simple construct. The estimate is the weighted mean of the two consecutive data points with the largest sum of weights. Complexity, assuming sorted data: $\mathcal{O}(n)$



- **MSV, M3V** The "maximal single value", and "maximal three values" – same as MTV, but with one or three values, respectively, instead of two.
- **MAMF** The "maximum area mode finder" – Same as the MTV, but respecting the weights, also ("area" comes from considering the weights as a second "dimension" of the data).

3.3 The "Clustering1D" package

Mode finding in one dimension in ORCA is implemented in a fully templated fashion; this enabled us to use the same algorithms for different applications. Apart from the use shown

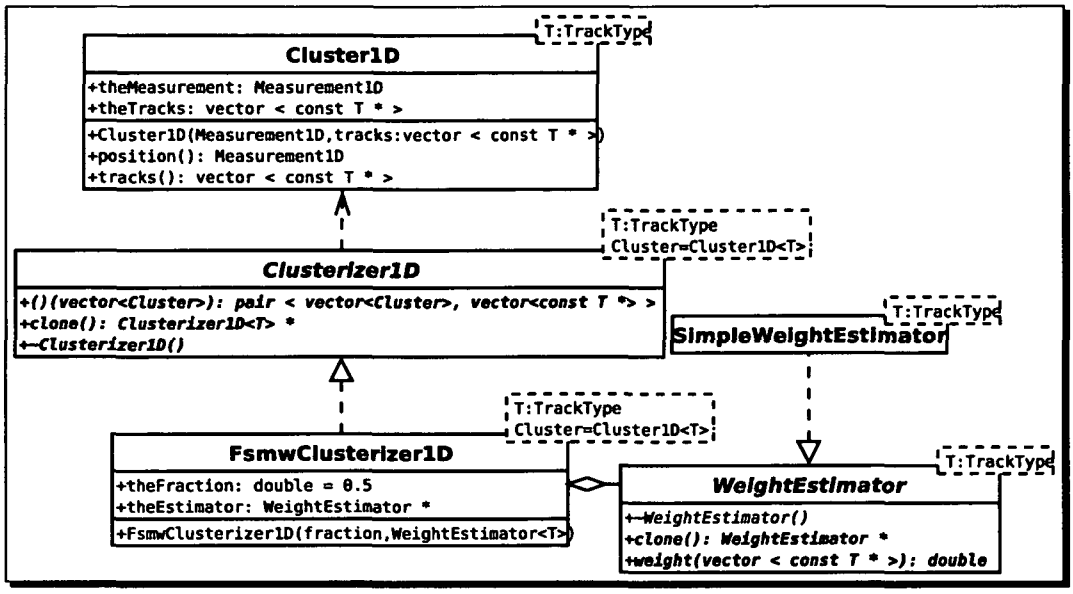


FIGURE 3.1: The Clustering1D design.

in this thesis (apex point finding, linearization point finding), the Clustering1D is also used in the context of primary vertex finding with pixel tracks [41]. The templated design of the package is depicted in Fig. 3.1. Central in this framework is the notion of Cluster1D<T>, T being the class of the "track" objects. This class consists of a Measurement1D — the position of the cluster, and a vector of tracks. This Cluster1D<T> class serves both as the input and the output of a *Clusterizer1D<T>*. In order to have maximum generality, *WeightEstimators* were introduced that return the weight associated to a Cluster1D<T>, depending on the tracks associated with the cluster.

Verification

The templated nature of the Clustering1D package makes code verification particularly fast and easy. The package's test directory has one simple program (ClusteringTest.cpp) that performs a few simple tests against the Clusterizer1D<char> algorithms. The only class that is external to the package, is Measurement1D; The test program thus compiles and runs very quickly, and is extremely insensitive to changes in the rest of ORCA.

3.4 Mode finding in three dimensions

We distinguish between coordinate-wise and "full" 3d mode finders, a coordinate-wise mode finder being an algorithm that splits up an n-dimensional problem into n one-dimensional problems. For the coordinate-wise mode-finders we have to bear in mind the following

unpleasant property:

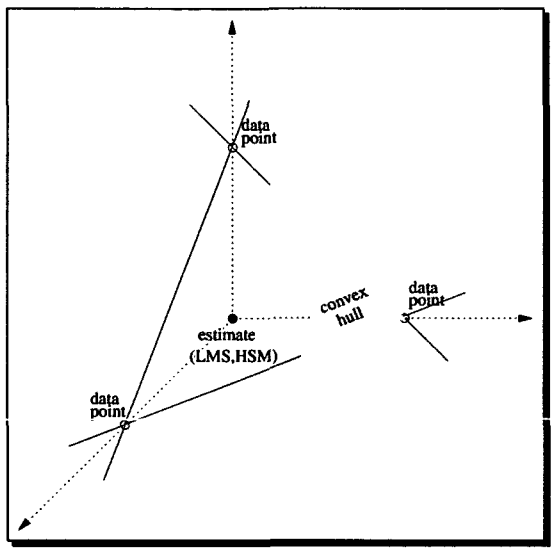


FIGURE 3.2: Illustration of the proposition.

Proposition 1. *The estimate of the coordinate-wise mode finders is in the general case not within the convex hull $\mathcal{H}$ of the data points.*

Proof. Consider the 3d input data $x_1 = (1,0,0), x_2 = (0,1,0), x_3 = (0,0,1)$. Since a mode finder is *defined* to return the most frequently observed data value, any coordinate-wise mode finder has to return: $\tilde{x} = (0,0,0) \notin \mathcal{H}$ □

This is a result with quite drastic consequences for us. Mode finding in higher dimensions turns out to be a very CPU-intensive task. So the above proposition seems to leave us only with the option of trading in quality for speed in case of n-dimensional mode finding problems.

Note, on the other hand, that in the case of highly collimated tracks the data points are aligned very close to a line. In that special case the above proposition loses some of its power. (The proposition does not hold in the special case that all points are aligned on a straight line).

"True" 3d mode finders

In the category of "true" 3d mode finders we only have a single family of algorithms:

- **SMS** Small Median of Squares: For each data point the median of the distances to all other points is computed. The data points are sorted according to their associated median distance. The mode is now the mean of the top n % of the data points, n being a quantity defined by the user.

- **ISMS** Iterated Small Median of Squares: Same as the SMS algorithm, only performed iteratively. It is analogous to the HSM algorithm in one dimension.
- **ISMSW** Iterated Small Median of Squares with Weights. Same as the ISMS, but the data points have now an associated weight. We associate a weight with the distance between two data points that is the sum of the weights of the data points. For each point the distances are, again, sorted according to their length. Starting from the top of the list we now add up the distances' weights until we exceed half of the total sum of all weights. This is now considered the "weighted average distance"; the data points are sorted by this quantity. The final estimate is now the weighted mean of the n % of the data points..

Literature knows of algorithms that try to minimize the LMS criterion [84], but they seem to be too CPU intensive for our needs.

3.5 Mode finding in 3d in ORCA

Mode finding in three dimensions is not as generic as the Clusterizer1D interface. It does not exploit an template technique and uses GlobalPoints as its "fundamental data type". The abstract interface is depicted in 3.3. Currently a few coordinate mode finders (FSMWModeFinder3d, HsmModeFinder3d, LmsModeFinder3d), as well as a few "true 3d" algorithms (SMSModeFinder3d) are implemented.

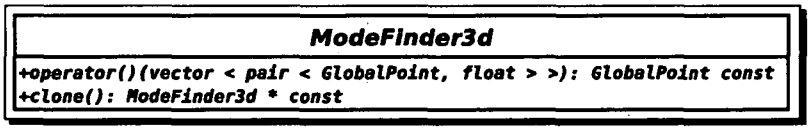


FIGURE 3.3: Purely abstract base class for mode finding in 3d

CHAPTER 4

Vertex Finding

Je ne cherche pas; je trouve.

P. Picasso

This chapter is devoted to the task of sorting a set of tracks into subsets that share points of origin. Both the mathematics and the implementations of different clustering algorithms are presented.

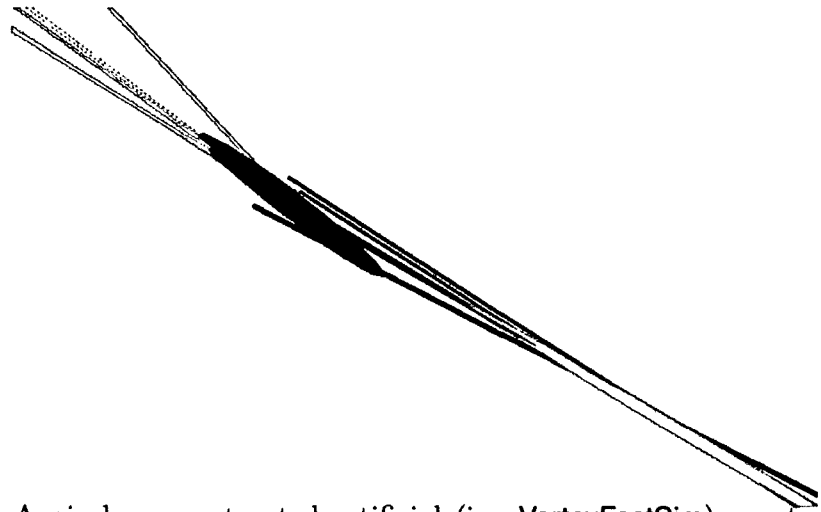


FIGURE 4.1: A nicely reconstructed artificial (i.e. VertexFastSim) event with a primary and a secondary vertex. Both vertices are found — the cyan and red ellipsoids represent RecVertex objects.

4.1 Introduction

It is the aim of this chapter to discuss vertex finding algorithms that operate on geometric knowledge only. Only the last section (Sec. 4.8) will briefly discuss general ideas of how to incorporate special physics knowledge into the geometric algorithms. The primary benchmark test shall be b-tagging. In this case the decay vertex of a B meson is to be found, with an average distance of ≈ 2 mm to the primary vertex, and an average track multiplicity of 4–5. Since the b-tagging efficiency in itself is not meaningful without taking into account fake rates, we compare results in bg topologies, with those in $c\bar{c}$ and $q\bar{q}$ topologies. This makes 2d efficiency versus fake rate plots possible.

Section 4.2 will introduce a broad categorization into the sea of algorithms, section 4.3 will say a few words about the input of vertex reconstructors; the apex point concept is mentioned here. Sections 4.4 and 4.5 will describe hierarchic and non-hierarchic methods, respectively. Section 4.6 will introduce *meta-algorithms*, i.e. vertex reconstructors that work on the input of two or more other vertex reconstructors. Finally, section 4.7 will describe methods used in other experiments that have not (yet) been considered for CMS, while section 4.8 will mention a few general ideas of how to insert specific physics knowledge into existing geometric algorithms. Comparisons in realistic use cases will not be given in this chapter — this task is delegated to Ch. 6.

4.2 Categorization

In order to strengthen the structure of this chapter, a fundamental categorization of algorithms shall be introduced. We shall distinguish between hierarchic and non-hierarchic algorithms; a hierarchic algorithm is one that can be visualized with a dendrogram (Fig. 4.5). Non-hierarchic methods are those that lack such a "dendroidal" representation of the procedure. Hierarchic methods can further be subdivided into agglomerative and divisive clusterers, depending on the "direction of operation" in the dendrogram. Divisive clusterers start with one big cluster that is then subdivided into sub-sets; agglomerative clusterers start with singleton groups — groups with one track only — that are then merged iteratively. Hierarchic algorithms are presented in section 4.4, with an agglomerative (4.4.1) and a divisive (4.4.2) subsection. Section 4.5 is dedicated to non-hierarchic algorithms.

4.3 Input data

A vertex reconstructor maps $\mathsf{vector} < \mathsf{RecTrack} > \rightarrow \mathsf{vector} < \mathsf{RecVertex} >$. Its input is a vector of **RecTracks**, its output is a vector of **RecVertices**. Distances between tracks feature unpleasant metric properties (see subsection 4.4.1). To circumvent these nasty characteristics, apex points have been introduced. The idea is straightforward: simple points in Euclidean 3d space are introduced representing the tracks. Being ordinary points in 3d space, the pattern recognition algorithms can again work in a simple Euclidean space with all its nice properties. The rest of this section is dedicated to this apex point approach.

4.3.1 Apex Points

ăpĕx -ĭcĭs m. the top; esp. the top of the conical cap of the Roman 'flamines' , or the cap itself; hence any crown, tiara, helmet; fig., highest honor, crown; gram., the long mark over a vowel.

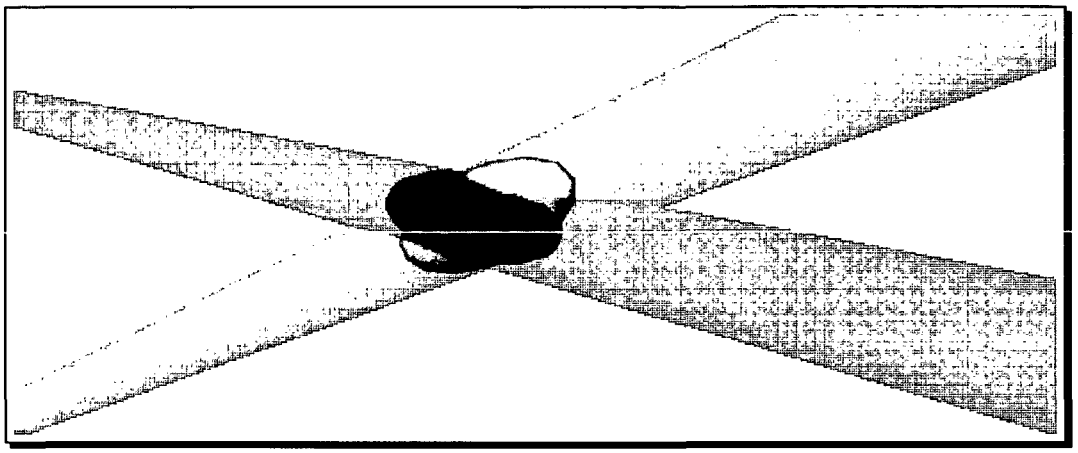


FIGURE 4.2: A $\mu^+\mu^-$ track pair, and the according apex points. Note how the apex points "inherit" the tracks' transversal errors. (Tracks are part of a simulated $J/\psi\ \phi \rightarrow \mu^+\mu^- K^+ K^-$ decay).

As has already been mentioned, the apex point formalism introduces representative points that live in 3d space. The formalism splits one pattern recognition problem in two: instead of finding a vertex from a set of tracks, this formalism finds points that represent the tracks, and only then a vertex from this set of representative points is searched for. Apex points serve as a nice play ground; since they are ordinary spatial points, one can implement practically any clustering algorithm that is known in literature. The apex point approach does not end with simple 3d spatial points; the track direction can naturally be introduced via a covariance matrix that is attributed to every apex point. The error perpendicular to the track direction is inherited from the track itself, the error longitudinal to the track direction is a property of the algorithm that finds the apex points (the ApexPointFinder); an infinite longitudinal error is equivalent to track linearization. Note that the distances weighted with the covariance matrices again do not form a metric space, as the triangle inequality can be violated. So, in some sense, the apex point formalism is merely a trade-off between track information and the "metricity" of the "feature space" (the space of the input data of the pattern recognition algorithms). Misery is thus conserved.

The mapping between tracks and apex points does not have to be unique: n apex points can be used to represent a track. Such an apex point finder is implemented; since it has never been seriously tested, it will not be mentioned in the rest of this section.

4.3.2 Apex point finders

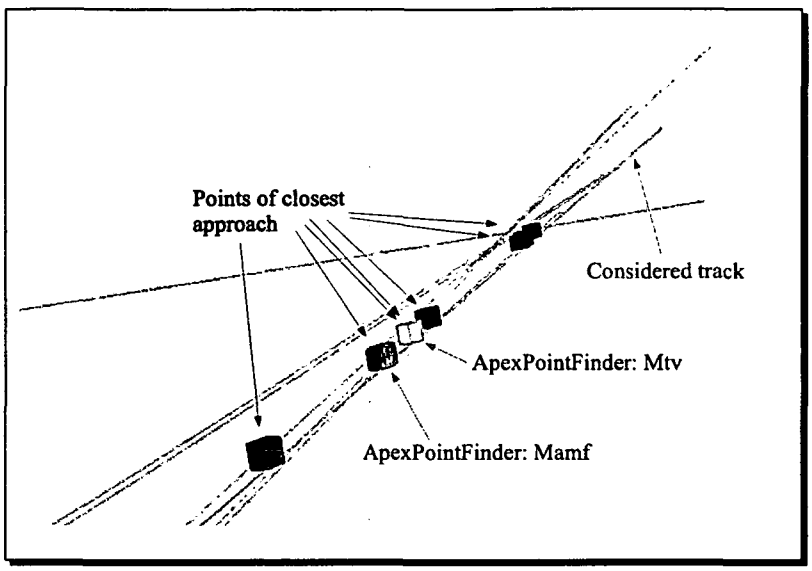


FIGURE 4.3: Two ApexPointFinders at work

An algorithm that searches for representative "apex" points is called an apex point finder; Fig. 4.3 visualizes the task. These finders operate on the points of closest approach (PtCAs); they determine the points of concentrations of these PtCAs. They can be formulated as a generic pattern recognition problem in one dimension (i.e. along the tracks). The information of the distance of the PtCAs to the track is exploited as an apex point "weight". Within ORCA, Clusterizer1D<RecTrack> (see Sec. 3.3) algorithms perform the task; input and output are the arc lengths of the points along the track, and their "weights". The latter are a function of the distances of the other tracks with respect to the track considered.

4.3.3 Analysis of the apex point finders

The apex point finders were compared with one another in a set of 1000 $b\bar{b}$ events. The AgglomerativeApexVertexReconstructor was used, with the various apex point finders (see Fig. 4.4). Comparing the "geometric scores" (see Sec. B.1.1) has the "MTV" algorithm appear as the most promising apex point finder. It is therefore established as the default method for the remainder of the thesis.

4.3.4 Apex fitting

Many issues relevant for vertex fitting are also relevant for apex fitting. Specifically, we can also formulate a linear fitter, and robustified methods, in full analogy with the vertex fitters.

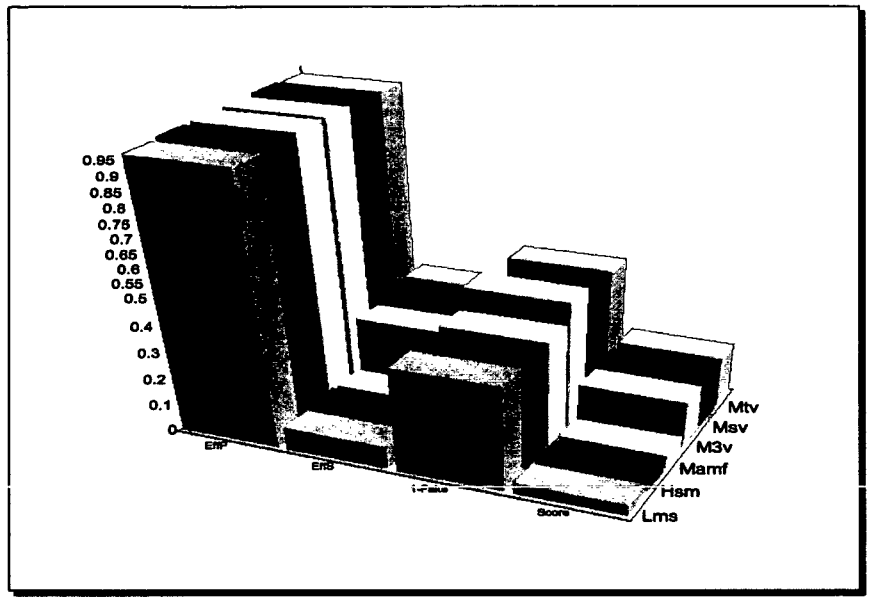


FIGURE 4.4: Comparison of different ApexPointFinder algorithms, with 1000(+300) $b\bar{b}$ events. "MTV" is the winner here.

The similarity between a least squares apex fitting method and the LinearVertexFitter is yet even greater. In fact, mathematically we can understand a LinearVertexFitter as a special *ApexFitter* with the following distinctive features:

- Its apex points are called points of closest approach. They have an infinite error longitudinal to the track. They minimize the distance to the linearization point.
- The whole fit is iterative, i.e. the apex points are not static but drawn towards the vertex candidate in every iteration step.

Apart from the above, they are mathematically equivalent. Thus, the robustification concepts of the vertex fitters (see Ch. 5) can be easily transferred to this framework. There is, in fact, also a very natural way of testing these new fitters. The only thing that is needed is a "fake" apex point finder that delivers apex points close to the SimVertex. This way, the apex fitter should give results that are statistically compatible with their VertexFitter counterparts.

One novel feature appears in the apex fitting approach that has no functional equivalent in the vertex fitting context: the quality attribute of the ApexStates, which can be used as a weight. This can be seen as a very simple robustification mechanism with virtually no CPU overhead. This will be the default robust apex fitting algorithm.

Summarizing it all up, we have now a

- LinearApexFitter

- AdaptiveApexFitter
- TrimmingApexFitter
- LinearApexFitter using the quality attribute as a weight.

The aforementioned fitters have been implemented in ORCA; testing has so far been very rudimentary only.

4.3.5 Performance limits of the apex point approach

It would be desirable to know the limits of the apex point approach; after all, it is an *ad hoc* idea that may or may not be useful. Such a study of its performance limits has not (yet) been performed. It will, though, be a necessity if one wants to come to a final decision about whether the apex point concept shall be pursued any further.

4.3.6 Final remark

The apex point formalism must definitely be put in question. It is an *ad hoc* idea that splits one pattern recognition problem in two. It has been introduced to deal with the fact that the triangle inequality is violated in the distance matrix between all tracks. The insertion of apex errors, though, reintroduced the "non-metric" aspect of problem, at least in principle, if maybe not so much in practice. To come to a final conclusion about the apex point concept one will have to study the performance limits of the approach. In order to overcome some of the limits of the approach, dynamic apex points – apex points that can move along the tracks - have been conceived. This will need highly specialized algorithms that are able to exploit the dynamic nature of the apex points. It will still take some more research until one can formulate a final decision on the whole concept.

4.4 Hierarchic Clusterers

4.4.1 Agglomerative clustering

Agglŏměro (adg-), -āvi, -ātum, 1, v. a., lit., to wind on (as on a ball); to add or join to, to annex; to join one's self to.

Agglomerative clustering algorithms start out with singleton groups, i.e. by assigning a separate group to every single track. The most compatible clusters are then iteratively merged until a stopping condition is met. The specific implementation is completely defined by its metric system, i.e. by its cluster-to-cluster compatibility criterion. We can distinguish between ORCA's various agglomerative clusterers in the following way:

(a) Agglomerative clustering in the distance matrix (AggloCAOH)

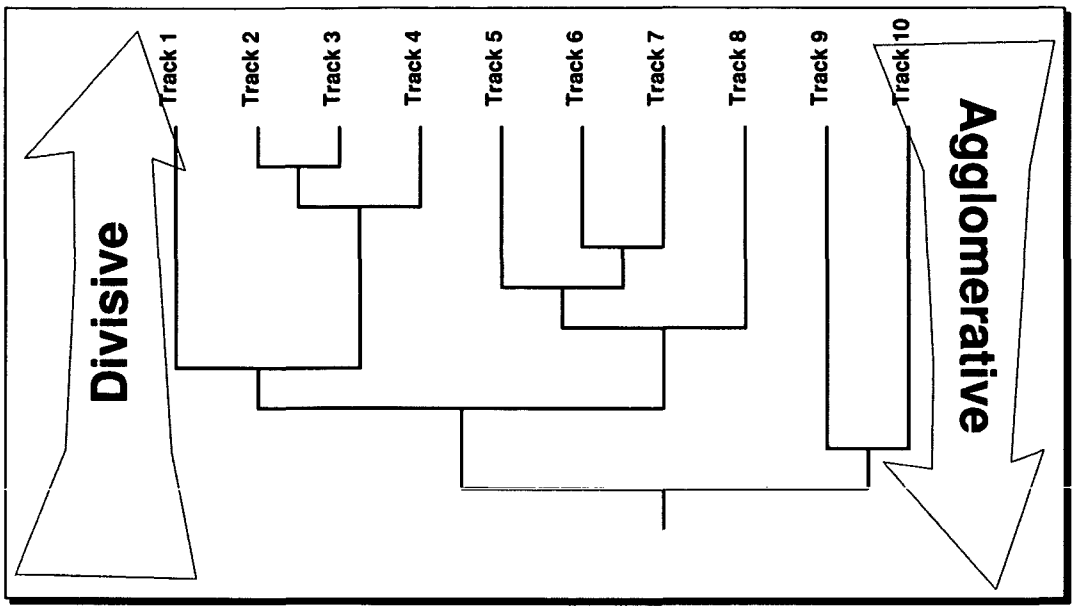


FIGURE 4.5: Every hierarchic clustering method can be visualized with such a dendrogram.

(b) Agglomerative clustering with fitted vertices as a "representative" of a cluster (AggloIP)

(c) Agglomerative clustering in the *ApexSpace* (AggloApex)

Let α and β denote two clusters. Let further s be the set of all minimum distances between track pairs with one track in cluster α and the other in cluster β. We can now choose as the metric e.g.:

$$d(\alpha, \beta) = \min(s), \max(s), \bar{s}, \text{median}(s), \ldots \tag{4.1}$$

The choice $d(\alpha, \beta) = \min(s)$ implements a *single linkage* or *minimum spanning tree* procedure, whereas $d(\alpha, \beta) = \max(s)$ is often referred to as a *complete linkage*.

The following proposition significantly reduces the number of reasonable choices:

Proposition 1. *The triangle inequality does not generally hold for the minimum distances between a set of n tracks.*

Proof. Let A, B, C denote three tracks. Let A and B share one common vertex V_1; let further B and C also share one common vertex V_2. Then:

$$\begin{aligned} &\overline{AB} = \epsilon, \overline{BC} = \epsilon, \overline{AC} = d \gg \epsilon \\ &\rightarrow \overline{AB} + \overline{BC} \ll \overline{AC} \quad \square \end{aligned} \tag{4.2}$$

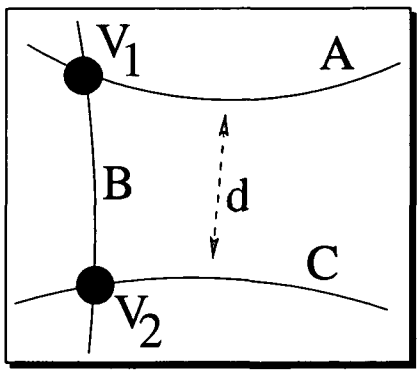


FIGURE 4.6: Schematic description of how the triangle inequality is violated in the track clustering problem.

This means that the choice $d(\alpha, \beta) = \min(s)$ would cluster A, B and C into a single vertex. We can therefore safely discard single linkage from the list of promising algorithms.

Until now the best results were obtained with the choice $d(\alpha, \beta) = \max(s)$, i.e. with a *complete linkage* procedure.

Implementation

The agglomerative clusterers have a package on their own: **AgglomerativeVertexReco**. All three types of agglomerative clusterers (clustering with representatives, clustering in the distance matrix, clustering in the apex space) have been implemented. The user can also specify a **StoppingCondition**. That way it is possible to implement a clusterer that reconstructs e.g. at least two vertices. In the case of clustering with representatives it is also possible for the user to provide the ***VertexFitter*** that is used to estimate the location of the representative point.

Implicit assumption

In order to be able to find a vertex, at least two tracks must be sufficiently compatible with one another. Both tracks must not be more compatible with another track than with the "partner" track. Compatibility is defined by the specific metric that is employed.

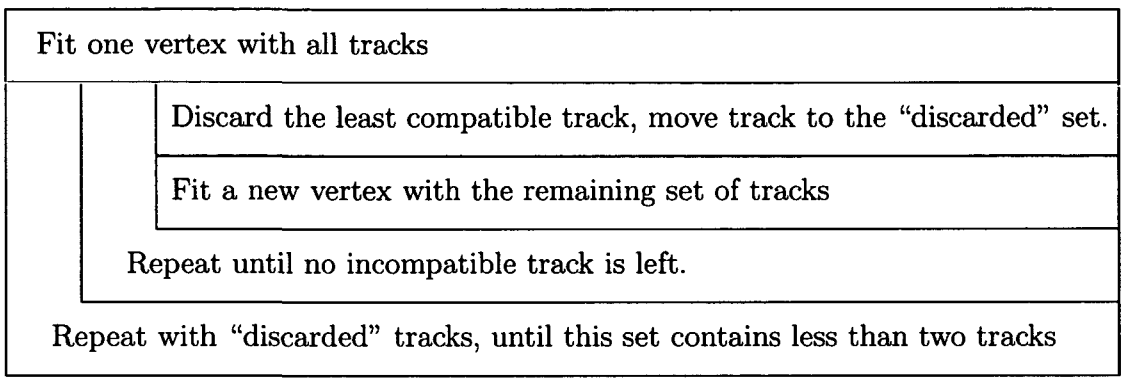
4.4.2 Divisive Clustering

Divisive clusterers start by assuming all tracks to be in one, huge cluster. This cluster is then further divided into smaller sub-clusters, which are then sub-divided until a formal criterion is met. ORCA has two divisive clusterers. Both work in a very similar manner: they fit a given set of tracks with a ***VertexFitter***, then discard the incompatible tracks, until

all tracks are considered compatible. The procedure is then repeated with the remaining set of tracks that are incompatible with the previous vertices. This procedure is also called "finding-through-fitting".

The principal vertex reconstructor

The PrincipalVertexReconstructor (PVR) has been developed in Brussels; given a vertex fitter, it is the most straightforward type of vertex finding algorithm. The PVR has been the first implementation of a vertex finder; ever since then it has served as a solid and stable baseline for all other finders. Its algorithm is very simple:



Implementation The PVR has its own package: PrincipalVertexReco. Its two most important parameters are:

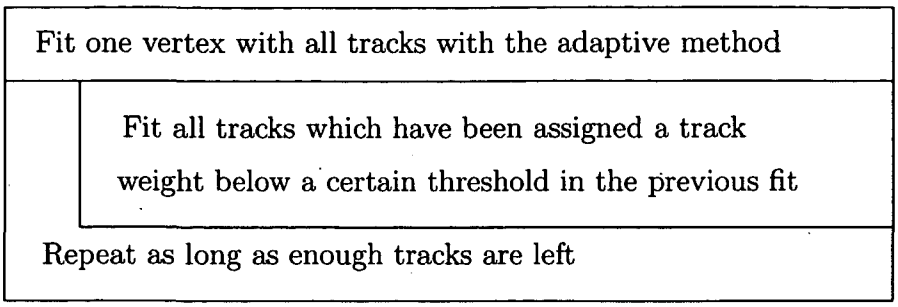
- the cut on the track probability — i.e. how *unlikely* does it have to be for a track to belong to a vertex for it to be discarded.
- the cut on the vertex fit probability: how improbable must a vertex fit be in order for the vertex to be discarded in a final cleaning step.

Recently, an experimental feature has been added that allows the cut on the track probability for the first (i.e. the primary) vertex to differ from the track probability cut from all the subsequent vertices.

Implicit assumption For a secondary vertex to be found, at least two tracks must be incompatible with the primary vertex, and they must be sufficiently compatible with one another. Compatibility is defined by the VertexTrackCompatibilityEstimator.

Adaptive vertex reconstruction

The AdaptiveVertexReconstructor (AVR) is also a very simple vertex reconstructor. Given an AdaptiveVertexFitter (Sec. 5.4.2), it can be described as:



Implementation Despite its inherent simplicity, the AVR was first implemented by the author only a few weeks before this thesis was turned in. At the time of this writing, it is not even in ORCA's CVS repository yet, although it certainly will be soon. Its implementation is trivial: An AVF is called iteratively.

Implicit assumption A secondary vertex can only be found if at least two of its tracks are incompatible with the primary vertex. ChiSquareForWeightComputation is responsible for the computation of the compatibilities.

Future development Many ideas that have been tried with the PVR can also be adapted to the AVR. Particularly, changing the χ^2_{cut} parameter after having fitted the primary vertex, is very interesting. Using soft constraints on the track weights, which has also been suggested for the multi vertex fitter (Sec. 5.5.3), is a viable option for the future, too.

4.5 Non-hierarchic clusterers

Non-hierarchic clusterers distinguish themselves by the lack of a strict clustering hierarchy. Most methods here employ the notion of mobile prototypes that are attracted to the data points. This concept has already appeared a few times in literature [71, 61]. The last algorithm presented in this section has no moving prototypes. Rather the data points are associated with an integral number which can be interpreted as a "cluster number". A generalization of such an association number would be a complete list, assigned to every data point, that keeps track of assignment probabilities of all data points to all clusters. This concept, too, has already appeared more than once in literature [25, 28]. The MultiVertexFitter that will be presented in Sec. 5.5 also functions in a similar way.

4.5.1 Vector quantization

Vector quantization [61] has been introduced as a method of "lossy" data compression; the data vectors are replaced by representative "code vectors", that are taken from a "code book". The code vectors need not live in the same space as the data vectors. Kohonen's

renowned self organizing maps (SOM) [71] are an example for such a vector quantization algorithm. The optimization of the code book determines the quality of the assignment.

In the vertexing context, the apex points have been chosen as the data vectors. The code vectors also live in the apex space; they are attracted by the apex points. The final position of the code vectors will provide the location of the vertex candidates. The ORCA implementation of a vector quantization algorithm is a "frequency-sensitive competitive learning" algorithm which means that:

- Iterating over all data vectors, every data vector "attracts" the closest code vector and pulls the code vector (the "vertex prototype") towards itself (competition),
- To make sure that all prototypes learn to represent data vectors, a "conscience" is introduced: code vectors which "win" frequently, are penalized in order to make way for the "losers" (frequency-sensitivity).

Clearly, the learning procedure relies on a distance measure between the apex point (data vector) and the prototype (code vector). Since we are dealing with Euclidean 3d space, the geometric distance between the two points is certainly an option. Another possibility takes into account the fact that the apex points come with a covariance matrix.

4.5.2 Weighted versus non-weighted learning

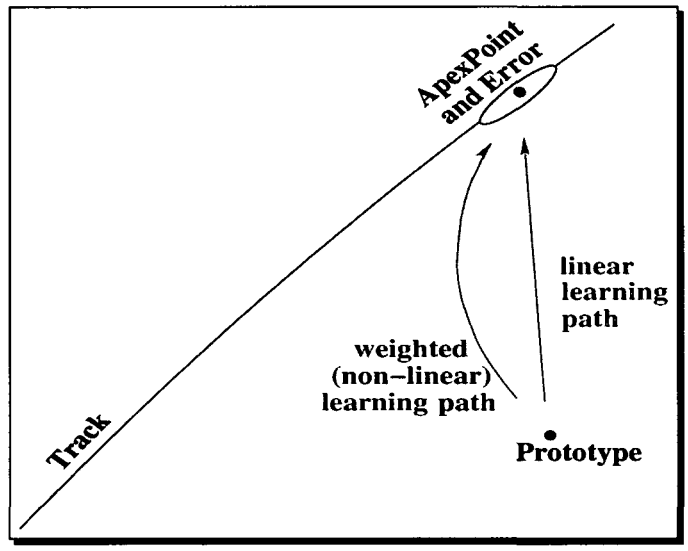


FIGURE 4.7: Weighted vs non-weighted learning

In order to account for the knowledge of the ApexErrors, a non-linear learning method is introduced. The distance vector ApexPoint – prototype is multiplied with the weight of the ApexPoint:

$$d(\vec{a},\vec{p}) \equiv (\vec{a}-\vec{p})\cdot \mathbf{W}_{\vec{a}} \tag{4.3}$$

where $\vec{a}$ is the position of the ApexPoint, $\vec{p}$ the position of the prototype and $\mathbf{W}_{\vec{a}}$ is the inverse of the covariance matrix of the ApexPoint.

What do these new "learning paths" look like? In our specific application the error in the direction of the track is usually significantly larger than the error perpendicular to the track direction. Hence, the prototypes are not directly drawn towards the ApexPoints, but more towards the tracks they represent. If one wants a more rigorous understanding of the non-linear "learning paths", we can without sacrificing generality consider a diagonal weight matrix [1]. For simplicity we consider a two dimensional example; the generalization to three dimensions is trivial. If $\vec{p}(t)$ denotes the path of the prototypes, $\vec{a}, \mathbf{W}_{\vec{a}}$ being the ApexPoint and the ApexWeight, then:

$$\frac{\vec{p}(t+\epsilon) - \vec{p}(t)}{\epsilon} = \mathbf{W}_{\vec{a}} \cdot (\vec{a} - \vec{p}(t)) \tag{4.4}$$

With $\epsilon \longrightarrow 0$ (i.e. infinitesimal learning steps) this leads to

$$\frac{d\vec{p}(t)}{dt} = \mathbf{W}_{\vec{a}} \cdot \vec{a} - \mathbf{W}_{\vec{a}} \cdot \vec{p}(t) \tag{4.5}$$

Without loss of generality we can consider the case of a diagonal weight matrix:

$$\mathbf{W}_{\vec{a}} \equiv \begin{pmatrix} W_{xx} & \\ & W_{yy} \end{pmatrix} \tag{4.6}$$

$$\dot{p}_x(t) = -W_{xx} \cdot p_x(t) + W_{xx} \cdot a_x \tag{4.7}$$

$$\dot{p}_y(t) = -W_{yy} \cdot p_y(t) + W_{yy} \cdot a_y \tag{4.8}$$

The solution is

$$p_x(t) = C_x \cdot e^{-W_{xx}t} + a_x \tag{4.9}$$

$$p_y(t) = C_y \cdot e^{-W_{yy}t} + a_y \tag{4.10}$$

$$\tag{4.11}$$

which defines $C_x = p_x(0) - a_x$. After a few trivial manipulations we arrive at:

$$p_y(t) = (p_y(0) - a_y) \cdot \left(\frac{p_x(t) - a_x}{p_x(0) - a_x} \right)^{\frac{W_{yy}}{W_{xx}}} + a_y \tag{4.12}$$

Extending to three dimensions will simply yield 4.12 again, only with z substituted for all occurrences of y.

4.5.3 The KMeans algorithm

The KMeans algorithm works similarly to the vector quantization, only the update of the prototypes is done in a batched fashion; all apex points are associated with one and only one prototype. The new position of a prototype is then the result of an "apex fit" (see Sec. 4.3.4), either linear or robust.

[1] One can always rotate into the eigensystem, then rotate back.

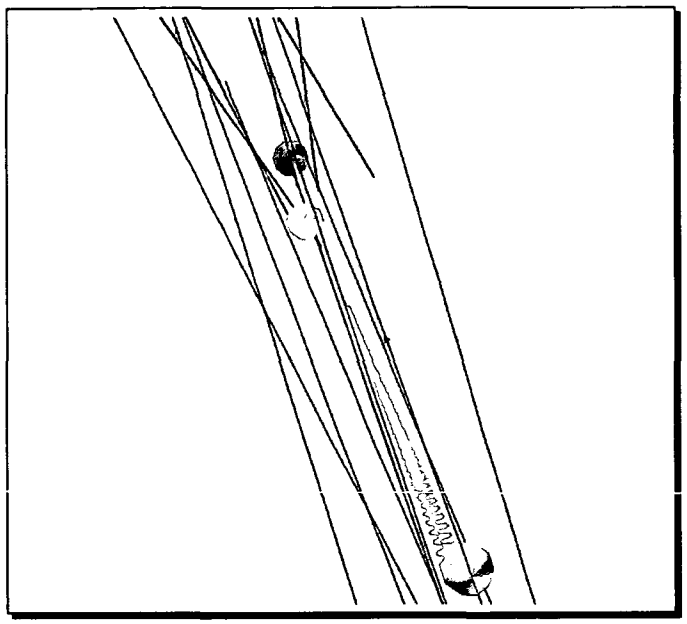


FIGURE 4.8: A prototype (initial position: gray sphere at the bottom) "learns" in a zigzag (weighted learning) towards the simulated vertex (yellow sphere) which has the red RecTracks attached to it.

4.5.4 Seeding

Ut sementem feceris, ita metes.

Marcus Tullius Cicero, De Oratore (II,65)

The KMeans and the VQuant algorithms need "seeding"; they need initial prototypes placed at some meaningful initial locations. A class that provides such information, in the form of a list <GlobalPoint>, is a *VertexSeeder*. Currently only the VQuantInitialiser is used: its central piece is a TrimmingApexFitter, which fits a prototype with a small fraction (typically some 25%) of the data points. The points compatible with the prototype are then discarded from the set of data points, and the fitter is applied again. This procedure is iterated until no more data points are left.

Implementation

The VQuantVertexReconstructor and the KMeansVertexReconstructor share one package: VQuantVertexReco. They both use the same initializer class: VQuantInitialiser. The current implementation is entirely based on the apex point concept; a future development could introduce an additional abstraction with respect to the input data; RecTracks and distances between them could just as well serve as input.

4.5.5 Deterministic annealing

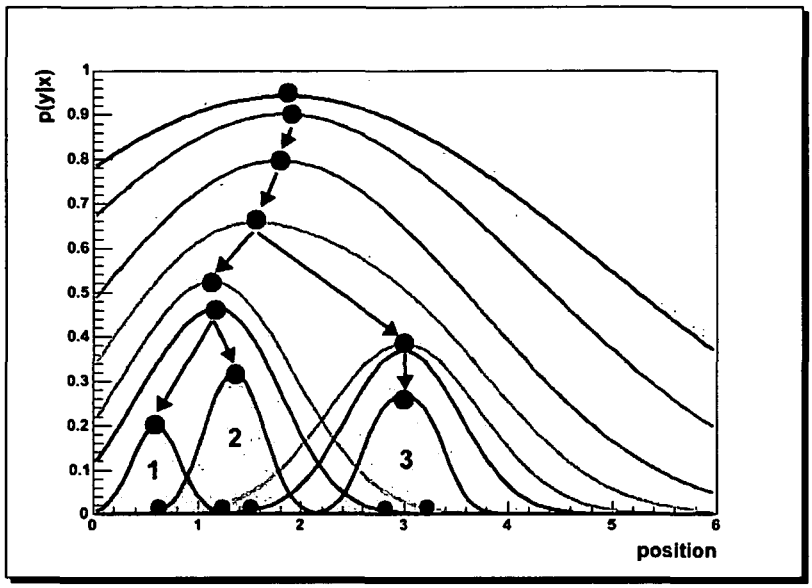


FIGURE 4.9: The deterministic annealing process (picture taken from [47])

The deterministic annealing algorithm [82, 83] is another method that exploits a physics analogy for the problem of finding natural clusters in data sets. This method starts with one "prototype" situated e.g. in the center of mass of the data points. In an annealing procedure the energy of the data points interpreted as a thermodynamic system is lowered. Every phase transition that the system goes through, a prototype splits up in two, along the principal component of the data associated with the prototype, see Fig. 4.9. A vertex reconstructor for CMS based on this concept has been developed in Lyon [48]. It is also based on the apex points. "Geometric" comparisons between this reconstructor and the PVR have been made [47]; performance comparisons in the context of b-tagging are still missing.

4.5.6 Super-paramagnetic clusterer

An inhomogeneous ferromagnet exhibits an intermediate state between the ferromagnetic and the paramagnetic phase. This state is commonly referred to as the super-paramagnetic phase. The super-paramagnetic clusterer(SPC) [28] exploits this analogy for clustering tasks. A spin equivalent is assigned to each apex point. The spin will interact and arrange themselves depending on a virtual temperature parameter, which is lowered over time. At a certain temperature the spins will align in typical ferromagnetic grains – the "Weiß regions". These strongly correlated regions are then identified as a cluster. An implementation of the algorithm in ORCA exists. No results have yet been obtained.

4.6 SuperFinders

A SuperFinder is a *VertexReconstructor* that internally relies on other VertexReconstructor implementations, and then combines the found solutions into one, hopefully better, solution. Combining solutions may be as simple as choosing one that minimizes a formal criterion, such as the global association criterion (see Sec. 4.6.1); it may also be a true combination, i.e. a new solution, such as the ones delivered by the voting algorithm described in Sec. 4.6.3.

4.6.1 Global association criterion (GAC)

The weights that have been introduced for the adaptive fitting method (see Sec. 5.4) can also be used to define a global "plausibility" criterion of the result of a vertex reconstructor. With m being the total number of tracks and n as the number of vertices we define the GAC by:

$$p = \frac{1}{n \cdot m} \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} + \lambda \cdot n \tag{4.13}$$

where

$$p_{ij} = \begin{cases} 1 - w_{ij} & \text{if } i \in j \\ w_{ij} & \text{otherwise} \end{cases} \tag{4.14}$$

and w_{ij} is the weight w_i (5.4) of track i with respect to vertex j. λ is an additional, optional parameter that can be used to penalize an excessive number of vertices.

The potential uses of such a criterion are manifold:

- Exhaustive vertex finding algorithm. All combinations of track clusters could at least in principle be iterated through, then one can decide for the smallest GAC found.
- Stopping condition. The GAC could also serve as a stopping condition in a wide range of algorithms.
- Super finder algorithms. One could also use it to resolve ambiguities. More than one vertex reconstructors could be used on one event, the GAC could then decide for the "better" solution.

The weights w_{ij} in Eq. 4.14 are a function of the cutoff parameter χ^2_{cut}. A lower limit on this cutoff parameter can be motivated by a purely analytic argument: let us assume one single vertex with n associated statistically correct tracks. The weight of each track with respect to the single vertex is

$$w_i\left(\chi^2\right) = \frac{1}{1 + e^{\frac{\chi^2 - \chi^2_{\text{cut}}}{2 \cdot T}}} \tag{4.15}$$

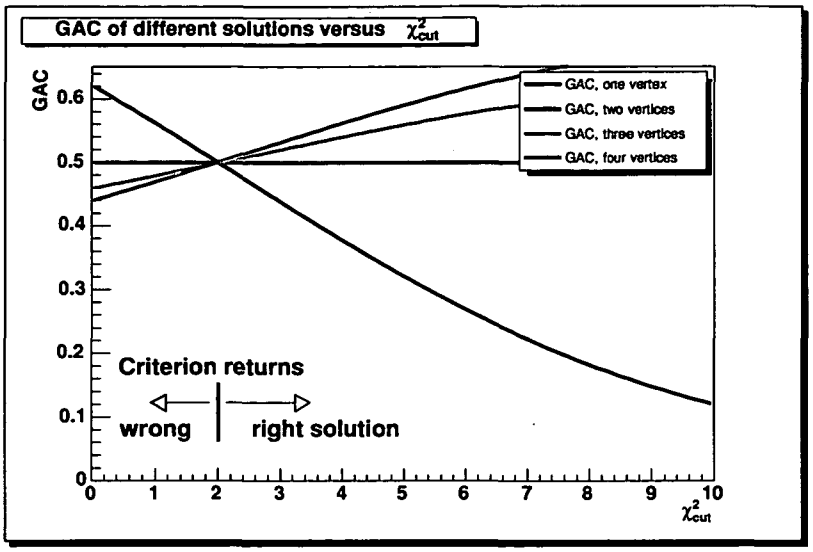


FIGURE 4.10: The GAC for different hypotheses (i.e. different clustering solutions), with $\lambda = 0$. The event contains one "perfect" vertex. Lower GAC corresponds to a more plausible hypothesis.

Taking the average of the w_i is in good (linear) approximation [2]

$$\overline{w_i} = \frac{1}{1 + e^{\frac{2-\chi^2_{cut}}{2 \cdot T}}} \tag{4.16}$$

This results in an average GAC for a single vertex hypothesis:

$$\overline{p}|_{1 \text{ vertex}} = \frac{1}{n} \sum_i (1 - \overline{w_i}) + \lambda = 1 - \overline{w_i} + \lambda \tag{4.17}$$

Let us now consider the hypothesis of two vertices, separated by $\epsilon \to 0$.
The average GAC for this hypothesis reads:

$$\overline{p}|_{2 \text{ vertices}} = \frac{1}{2n} \sum_i (p_{i1} + p_{i2}) + 2\lambda = \frac{1}{2} + 2\lambda \tag{4.18}$$

The same for three vertices is:

$$\overline{p}|_{3 \text{ vertices}} = \frac{1}{3} (1 + \overline{w_i}) + 3\lambda \tag{4.19}$$

And generally for m vertices, $m < n$:

$$\overline{p}|_{\text{m vertices}} = \frac{1}{n \cdot m} \cdot [n (1 - \overline{w_i}) + (m - 1) n \overline{w_i}] + m\lambda = \frac{1}{m} + \frac{m-2}{m} \overline{w_i} + m\lambda \tag{4.20}$$

[2] We shall see that the most interesting part of the approximation is around $\chi^2 = 2$. At this value the second derivative vanishes.

For the special choice of $\lambda = 0$, we have $w_i(2) = \frac{1}{2}$. This leads to $p_m(\chi^2_{\text{cut}} = 2) = \frac{1}{2}$, for *any* m! Thus, in order for this situation to be classified correctly (remember that the lower the GAC, the better), one will have to choose $\chi^2_{\text{cut}} > 2$ for any value of the temperature T (for $\lambda = 0$).

An open question

The most important open question with respect to this criterion is how it relates to the "Minimum Message Length" [100]. Can the information theoretic limit of the vertex finding task be formulated in terms of the GAC? It should also be investigated whether something can be gained by using the Akaike (AIC) or the Bayesian (BIC) information criterion rather than our GAC.

4.6.2 Best solution finder

The **BestSolutionFinder** is an experimental **SuperFinder** that uses the GAC to determine to most plausible result. A few tests have been made. Fig. 4.11 shows a comparison of the **BestSolutionFinder** with its constituting components. "Score" in this context refers to the geometric score function that is defined in Sec. B.1.1. The **BestSolutionFinder** does not exhibit the desired properties: the solution that is "best" according to the GAC is not always optimal by the "score" criterion. Despite it being a persuasive concept, development on GAC-based vertex finders has been suspended due to its lack of success.

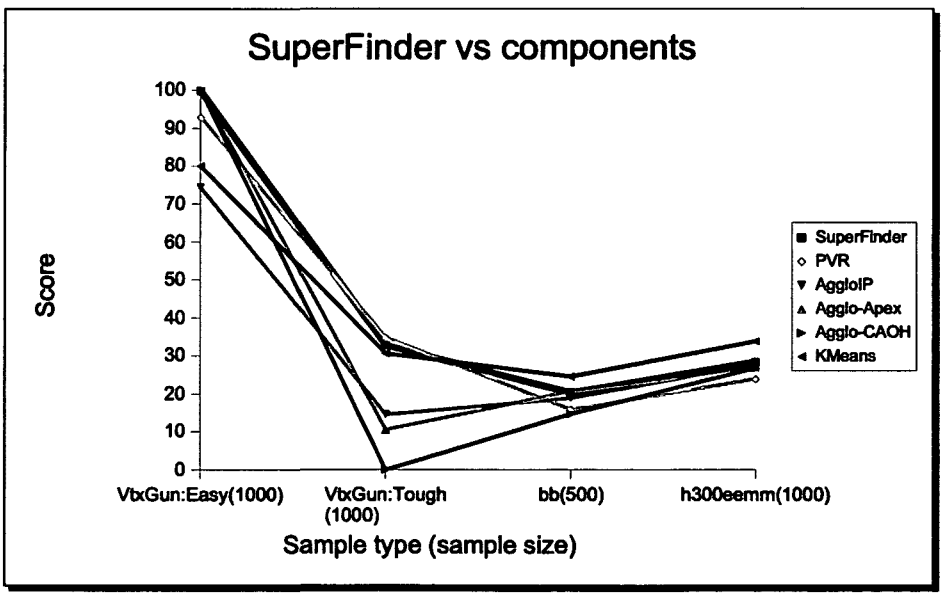


FIGURE 4.11: Comparison of the **BestSolutionFinder** with its constituting components.

4.6.3 Voting system

Another option for combining the results of different vertex finders into one single, hopefully better result has been suggested in [102].

Our (extended) version of the voting procedure can be described as follows: Given n solutions to a clustering problem, we start by picking two. We now try to identify the most similar clusters. The similarity of two clusters α and β is defined as a "relative" similarity:

$$sim\,(\alpha,\beta) \equiv \frac{p\,(\alpha \cap \beta)^2}{p\,(\alpha)\cdot p\,(\beta)} \tag{4.21}$$

where $p\,(\alpha)$ denotes the cardinality (i.e. the number of elements) of cluster α. Starting with the most similar clusters, we now construct the corresponding cluster for the new solution. If α_i, β_j denote the similar clusters in our two solutions, we now iterate through all elements in both solutions. Each element that appears in both clusters is added to our new cluster γ with a weight of one. Each element that appears in only one cluster is added with a weight of $\frac{1}{2}$. The two merged clustered are now "erased" from the set of clusters of the two original solutions. The procedure is now repeated: the two most similar clusters are searched for, then merged, then erased from the original sets. After having iterated through all clusters in both solutions, the "merged" solution $S_1' = S_1 \oplus S_2$. A third, or, more general, an nth solution can now be added in similar way. We only need to take into account that the cluster that carries the information of the previous $n-1$ solutions, receives a weight of $\frac{n-1}{n}$, while the "new" solution that we take into account receives a weight of $\frac{1}{n}$, n being the number of solutions considered so far, including the one that is in the process of being "added". The similarity criterion also has to respect the weights: If $\alpha^i \equiv (n_1^i A_1, n_2^i A_2, \cdots, n_k^i A_k)$ (n_1^i being the weights assigned to the elements A_1), then:

$$\begin{aligned}
p\,(\alpha^i) &\equiv \sum_{j=1}^{k} n_j^i \text{ ("cardinality")} \\
\alpha^i \cap \alpha^j &\equiv \left(n_1^i n_1^j A_1, n_2^i n_2^j A_2, \cdots, n_k^i n_k^j A_k\right) \text{ ("intersection")} \\
\alpha^i \ominus \alpha^j &\equiv \left([n_1^i - n_1^j]A_1, [n_2^i - n_2^j]A_2, \cdots, [n_k^i - n_k^j]A_k\right) \text{ ("subtraction")}
\end{aligned} \tag{4.22}$$

The description given above may be confusing; a simple example will make things clearer. A few simple definitions might help make the example more readable: Let S_1 be a "solution" that consists of the cluster $S_1 = (\alpha_1, \alpha_2, \cdots, \alpha_n)$. Let S_2 be another solution with the clusters $S_2\,(\beta_1, \beta_2, \cdots, \beta_m)$. We define:

Definition 1.

$$S_1 \odot S_2 = (\alpha_1, \cdots, \alpha_n, \beta_1, \cdots, \beta_m) \text{ ("concatenation")}$$

Example:

$$\boxed{AB} \odot \boxed{C}\boxed{D} = \boxed{AB}\boxed{C}\boxed{D}$$

Example 1. Consider the three "original" solutions:

$$\begin{aligned} S_1 &\equiv \boxed{AB}\boxed{CDEF}\boxed{G}\boxed{H} \\ S_2 &\equiv \boxed{ABC}\boxed{DEFG}\boxed{H} \\ S_3 &\equiv \boxed{ABE}\boxed{CD}\boxed{FGH} \end{aligned} \tag{4.23}$$

The voting schema now starts to merge S_1 and S_2:

$$S_1' = \frac{1}{2}S_1 \oplus \frac{1}{2}S_2 = \boxed{AB\tfrac{C}{2}}\boxed{\tfrac{C}{2}DEF\tfrac{G}{2}}\boxed{\tfrac{G}{2}}\boxed{H} \tag{4.24}$$

Or one could just as well start with the solutions S_2 and S_3:

$$S_2' = \frac{1}{2}S_2 \oplus \frac{1}{2}S_3 = \boxed{AB\tfrac{C}{2}\tfrac{E}{2}}\boxed{\tfrac{F}{2}\tfrac{G}{2}H}\boxed{\tfrac{C}{2}D\tfrac{E}{2}\tfrac{F}{2}\tfrac{G}{2}} \tag{4.25}$$

The third solution now has to be added:

$$S_{\text{final}}^{(1)} = \frac{2}{3}S_1' \oplus \frac{1}{3}S_3 = \boxed{AB\tfrac{C}{3}\tfrac{E}{3}}\boxed{\tfrac{2C}{3}D\tfrac{2E}{3}\tfrac{2F}{3}\tfrac{G}{3}}\boxed{\tfrac{G}{3}}\boxed{\tfrac{F}{3}\tfrac{G}{3}H} \tag{4.26}$$

Or, another legitimate way would be to:

$$S_{\text{final}}^{(2)} = \frac{2}{3}S_2' \oplus \frac{1}{3}S_1 = \boxed{AB\tfrac{C}{3}\tfrac{E}{3}}\boxed{\tfrac{2C}{3}D\tfrac{2E}{3}\tfrac{2F}{3}\tfrac{G}{3}}\boxed{\tfrac{G}{3}}\boxed{\tfrac{F}{3}\tfrac{G}{3}H} \tag{4.27}$$

Relieved, we note that $S_{\text{final}}^{(1)} = S_{\text{final}}^{(2)}$.

Resolving ambiguities

Special considerations have to be taken in situations when there is no unique pair of most similar clusters. Eq. (4.28) shows a simple case. Both clusters of S_2 have a similarity with the cluster of S_1 of $\frac{2^2}{2\cdot 4} = \frac{1}{2}$. Randomly choosing one cluster to be merged with the large "mother cluster" would introduce a bias; the symmetry that the "original" solutions exhibit would not appear in the final solution. Hence, a mechanism to resolve any such ambiguity is desirable. We demand of the resulting solution to show the same symmetries that exist in the initial solutions. The following mechanism has been conceived: Let α be one large "mother" cluster. Let further $\beta_{1,2,\ldots,n}$ denote the n "daughter" clusters with the same similarity with respect to their mother. The result that comes from associating daughter β_i with its mother α shall be denoted with R_i. E_i shall further be the solution R_i, only the big cluster that comes from "merging" the daughter with the mother shall be removed. M_i shall denote that cluster that has been removed: $R_i = M_i \odot E_i$. Now one specific "asymmetric" solution R_i is chosen. For simplicity let us choose R_1. Now the intersection of all $E_j, j \neq 1$ is computed: $I = E_2 \cap E_3 \cdots E_n$. I is now "subtracted" from M_1: $M_1' = M_1 \ominus I$. The final solution is then $M_1' \odot E_1$. "Bidirectional" ambiguities, as shown in 4.35, can be resolved with this mechanism as well, if only we ignore one ambiguity, i.e. we simply decide for any single "mother" cluster. Again the description is complicated, a few simple examples are needed:

Example 2. We start with a very simple example:

$$\begin{aligned} S_1 &\equiv \boxed{ABCD} \\ S_2 &\equiv \boxed{AB}\boxed{CD} \end{aligned} \tag{4.28}$$

The input is invariant under the permutation of any two tracks. This symmetry is supposed to appear also in the merged solution. Let us start by computing R_i, identifying M_i and E_i:

$$\begin{aligned} R_1 = \frac{1}{2}S_1 \oplus_{(asym,1)} \frac{1}{2}S_2 &= \underbrace{\boxed{AB\tfrac{C}{2}\tfrac{D}{2}}}_{M_1}\underbrace{\boxed{\tfrac{C}{2}\tfrac{D}{2}}}_{E_1} = M_1 \odot E_1 \\ R_2 = \frac{1}{2}S_1 \oplus_{(asym,2)} \frac{1}{2}S_2 &= \boxed{\tfrac{A}{2}\tfrac{B}{2}CD}\boxed{\tfrac{A}{2}\tfrac{B}{2}} = M_2 \odot E_2 \end{aligned} \tag{4.29}$$

Let us choose R_1 as the starting point for our "symmetrization technique". Now comes the intersection of all E_i:

$$I_1 = E_2 = \boxed{\tfrac{A}{2}\tfrac{B}{2}} \tag{4.30}$$

"Subtracting" the intersection from M_1:

$$M_1' = M_1 \ominus I_1 = \boxed{\tfrac{A}{2}\tfrac{B}{2}\tfrac{C}{2}\tfrac{D}{2}} \tag{4.31}$$

M_1' already looks nicely symmetric. As a last step we glue together all parts:

$$S_{\text{final}}^{(1)} = M_1' \odot E_1 = \boxed{\tfrac{A}{2}\tfrac{B}{2}\tfrac{C}{2}\tfrac{D}{2}}\boxed{\tfrac{A}{2}\tfrac{B}{2}}\boxed{\tfrac{C}{2}\tfrac{D}{2}} \tag{4.32}$$

The solution exhibits all the expected symmetries. Just to check, we compute $S_{\text{final}}^{(2)}$:

$$S_{\text{final}}^{(2)} = M_2' \odot E_2 = \boxed{\tfrac{A}{2}\tfrac{B}{2}\tfrac{C}{2}\tfrac{D}{2}}\boxed{\tfrac{A}{2}\tfrac{B}{2}}\boxed{\tfrac{C}{2}\tfrac{D}{2}} \tag{4.33}$$

Example 3. This example is a tiny bit more complicated.

$$\begin{aligned} S_1 &\equiv \boxed{ABCDEF} \\ S_2 &\equiv \boxed{AB}\boxed{CD}\boxed{EF} \\ S_1 \oplus_{(asym,1)} S_2 &= \boxed{AB\tfrac{C}{2}\tfrac{D}{2}\tfrac{E}{2}\tfrac{F}{2}}\boxed{\tfrac{C}{2}\tfrac{D}{2}}\boxed{\tfrac{E}{2}\tfrac{F}{2}} = M_1 \odot E_1 \\ S_1 \oplus_{(asym,2)} S_2 &= \boxed{\tfrac{A}{2}\tfrac{B}{2}CD\tfrac{E}{2}\tfrac{F}{2}}\boxed{\tfrac{A}{2}\tfrac{B}{2}}\boxed{\tfrac{E}{2}\tfrac{F}{2}} = M_2 \odot E_2 \\ S_1 \oplus_{(asym,3)} S_2 &= \boxed{\tfrac{A}{2}\tfrac{B}{2}\tfrac{C}{2}\tfrac{D}{2}EF}\boxed{\tfrac{A}{2}\tfrac{B}{2}}\boxed{\tfrac{C}{2}\tfrac{D}{2}} = M_3 \odot E_3 \\ I_1 &\equiv E_2 \cap E_3 \\ S_{\text{final}} &= (M_1 \ominus I_1) \odot I \odot E_1 = \boxed{\tfrac{A}{2}\tfrac{B}{2}\tfrac{C}{2}\tfrac{D}{2}\tfrac{E}{2}\tfrac{F}{2}}\boxed{\tfrac{A}{2}\tfrac{B}{2}}\boxed{\tfrac{C}{2}\tfrac{D}{2}}\boxed{\tfrac{E}{2}\tfrac{F}{2}} \end{aligned} \tag{4.34}$$

Example 4. It is left to the user to show that in the case below the merged solution does not depend on the choice of the "mother" cluster:

$$\begin{aligned} S_1 &\equiv \boxed{AB}\boxed{CD} \\ S_2 &\equiv \boxed{AC}\boxed{BD} \\ S_1 \oplus S_2 &= \boxed{\tfrac{A}{2}\tfrac{B}{2}}\boxed{\tfrac{A}{2}\tfrac{C}{2}}\boxed{\tfrac{B}{2}\tfrac{D}{2}}\boxed{\tfrac{C}{2}\tfrac{D}{2}} \end{aligned} \tag{4.35}$$

Order dependence

The whole procedure unfortunately is not order independent; a simple order dependent example is the set with the three solutions $\boxed{AB}\boxed{C}\boxed{D}$, $\boxed{ABCD}$, and $\boxed{A}\boxed{B}\boxed{CD}$. The procedure will introduce a bias towards the first solution. The ORCA implementation sorts the solutions by the number of clusters they contain, in ascending order. This minimizes the number of small clusters in the final solution.

Implementation

The ORCA implementation has been written in a fully templated fashion, the type of the fundamental elements can be defined by the user. The code has been verified (exploiting the templated nature of the code — it operates on simple ASCII characters rather than RecTracks). It should be very easy to recycle the code for any other application.

A VotingVertexReconstructor serves as the glue code between the templated VotingSchema and the vertex package. Any set of VertexReconstructors (including the VotingVertexReconstructor itself) can be specified as the primary source of solutions. The solutions are then combined in the way described in this section. The result is then fed to a MultiVertexFitter (see Sec. 5.5). The weights from the voting schema are used as the initial weights in the multi vertex fit.

Results

The b-tagging performance was compared between the voting vertex reconstructor and its constituents. In order to remove its influence, the same multi vertex fitter has been applied to the results of the constituents. The default values of the algorithms has been used. Fig. 4.12 shows the results. No clear performance gain is visible.

4.7 Other methods

ORCA has had one additional (secondary) vertex reconstructor, the "d0phi" method [86]. In this method the tracks are linearized at the points of closest approach to the primary vertex. Each track is then associated with a point in the $d_0 - \phi$ plane, where d_0 is the track impact parameter, and ϕ the azimuthal angle. Tracks which share a secondary vertex are roughly aligned in the $d_0 - \phi$ plane, with a positive slope, while tracks from the primary

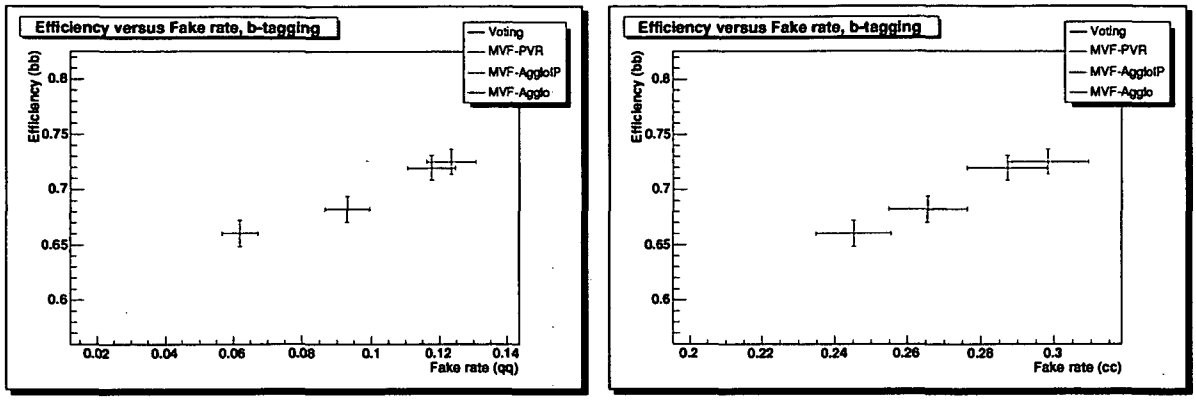


FIGURE 4.12: Comparison of the b-tagging efficiencies of the voting algorithm with its constituents.

vertex appear unaligned. The pattern recognition problem thus transforms into the task of finding straight lines in the $d_0 - \phi$ plane. An implementation exists in ORCA, but it is orphaned.

Literature names only a few more (geometric) vertex reconstruction techniques that have not yet been considered in CMS; the topological vertex reconstruction [69] seems especially appealing. Certainly, the number of general purpose clustering algorithms known to literature is immense. It is, though, the subjective feeling of the author that there is no need for any more pattern recognition algorithms. Future work should rather focus on how to improve upon the existing algorithms, what information should be fed to the existing non-hierarchic methods, etc.

4.8 Physics knowledge

In many use cases users will want to insert knowledge about the specific physics channels into the vertex finding procedure. The standard b-tagging application is a good example. Users will apply many different pieces of information to enhance b-tagging performance. A few good general ideas are known:

- **Ghost track formalism** [18] B mesons have a high probability of decaying into D mesons further "downstream"; for vertex reconstruction this poses a serious problem: instead of one vertex with ≈ 5 tracks, one ends up with two separate vertices and lower multiplicities. The *ghost track* has been conceived as a remedy: another "fake" track is introduced that points from the primary vertex in the direction of the B meson (i.e. in the jet axis). This "ghost track" can further help reconstruct secondary vertices — in theory it enables one to reconstruct 1-prong vertex decays! This idea is currently being implemented into ORCA by the b-tagging group.

- **Vertex suppression field** Some algorithms might be able to exploit some prior information of where a vertex is likely to be found and where vertices should be "suppressed". Such information could be passed to a vertex reconstructor in the guise of a scalar field. A possible use case is the suppression of vertices at the "fringes" and outside a jet cone.

CHAPTER 5

Vertex Fitting

"That's right," shouted Vroomfondel, "we demand rigidly defined areas of doubt and uncertainty!"

Douglas Adams, "The Hitchhiker's Guide to the Galaxy"

This chapter deals with the statistical problem of estimating the location and the error of a vertex. Dealing with finding a good initial guess at first, it then goes on to describe the well known least-squares techniques based on the Kalman filter. Least squares methods are non-robust; the chapter then discusses potential robustifications. Three robust algorithms are then implemented and compared with each other in well-controlled data sets. The chapter focusses on geometric properties only; special physics knowledge does not enter.

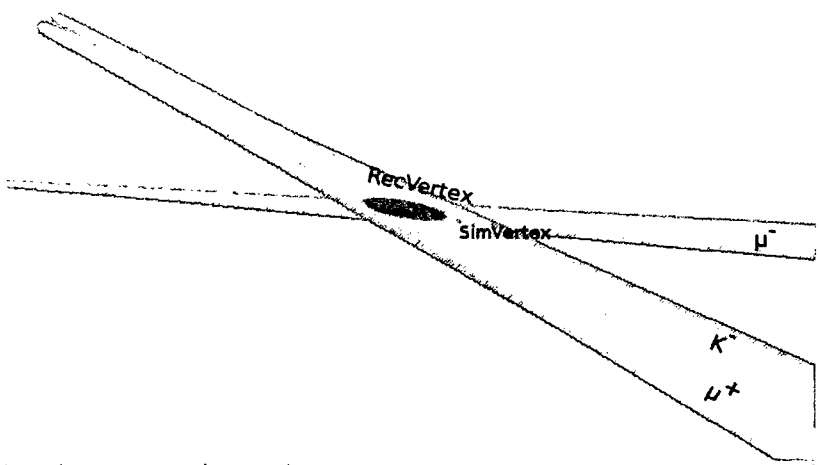


FIGURE 5.1: A $J/\psi\,\phi \rightarrow K^+K^-\mu^+\mu^-$ decay vertex, fitted with a linear fitter. K^+ was not reconstructed properly by the track reconstructor, therefore the track is missing.

5.1 Introduction

Vertex fitting deals with the "geometric" estimation of the location and the error of a vertex. This chapter will mainly look at Kalman filter techniques, which make it possible to sequentially add the information of a track to the vertex fit. Linearization of the tracks is required; the choice of parametrization, the specific track state, is not unique. In practice, though, it is restricted to only a very few possibilities. Sec. 5.3 will list two choices of parametrization. Linearization needs a point around which the tracks are linearized. This point is called a linearization point; Section 5.2 discusses a few approaches of how one can come up with a good initial guess. Linearization point finders will gain special importance in the context of robustifying the least squares methods. Robustifications will be presented in Sec. 5.4. The multi vertex fitter presented in Sec. 5.5 is a generalization of the adaptive estimator of Sec. 5.4. It can fit n vertices at once, and the vertices "compete" for the tracks. This method will be interesting when reconstructing entire decay chains. The subsequent sections deal with adding prior information (5.6), and an introduction of the Gaussian-sum filter (GSF, section 5.7). The implementation of the GSF is not part of this thesis, it is only mentioned because it can be combined with the adaptive method (5.8) and the multi vertex fitting method (5.9). This algorithmic chapter concludes with the derivation of the adaptive and the multi vertex algorithm from basic theoretical models (5.10).

5.2 Linearization point finders

All fitters, robust or not, require an initial rough guess of the vertex location. Such a guess is called "linearization point"; an algorithm that finds such a point is a *LinearizationPointFinder*. In the case of least squares fitters it merely determines the point around which the tracks are linearized. The least squares fitters are written such that too large a discrepancy between linearization point and vertex position triggers a re-linearization of all tracks. Thus, the initial linearization point should not play a crucial role. It should only show in the CPU cost of the fitter, not in the final result (at least not statistically). In combination with a robust fitter the initial guess plays a much greater role. In this case it represents the global aspect of the optimization problem. The robust fitters are but local optimization algorithms. A good initial estimate is thus crucial.

The section starts by sorting all implementations into various, general categories (5.2.1), before it discusses implementation details (5.2.2). It then describes how the performance can be analyzed (5.2.3) and shows the results (5.2.4). The next to last subsection (5.2.5) documents the importance of having a good linearization point finder. Finally, general conclusions are drawn (5.2.6).

5.2.1 Categorization

A *LinearizationPointFinder* is an algorithm that maps a vector < RecTrack > onto a GlobalPoint. We can distinguish between two broad categories of *LinearizationPointFinders* ac-

cording to how they "process" the input **RecTracks**: apex point based algorithms map the **RecTracks** onto **ApexPoints**, then fit the apex points with an **ApexFitter** (4.3.4). The crossing point based algorithms compute the crossing points (i.e. the algebraic mean of the two points of closest approach) of all track pairs. The crossing points then serve as input for a 3d mode finder (3.4).

Weighting the crossing points

The crossing point based algorithm can exploit not only the location of the crossing point. In addition a "quality" criterion can be assigned to the crossing points. This "weight" is a function of the distance of the two points of closest approach — the further apart the two PtCAs, the less a crossing point should "count". For linearization point finding the following general function has been used for weight assignment:

$$w = |d + d_{\text{cut}}|^n \tag{5.1}$$

d_{cut} and n are user-definable. Typically d_{cut} should be chosen to be of the same order of magnitude as the transversal track errors, n should be -1 or -2. Current defaults are $d_{\text{cut}} = 10\mu\text{m}$, $n = -2$. Fig. 5.2 depicts the weight functions for a few parameter choices.

Choosing a subset of crossing points

All crossing point based algorithms further have in common that the user can choose whether all possible track pairs (**n_pairs=-1**) or only a certain subset of track pairs should be considered. If the user-supplied **n_pairs** is positive then only a subset of crossing points is computed. The algorithm that chooses the subset is designed to benefit from the *ad hoc* assumptions:

(a) Tracks with higher p_T tend to be a "better" source of crossing points.

(b) Very high p_T tracks (e.g. in jets) tend to be highly collimated. It might be better to pair high-p_T tracks with tracks with a lower p_T, as long as one remains compatible with (a).

The algorithm that has been implemented fulfills these criteria. The simplest description is by example (Fig. 5.2). The performance of this sorting algorithm has never been analyzed. It is currently unknown if it performs better than a mere random generator! Anyway, the CPU cost of this sorting is very low and the idea of introducing an "upper combinatorial" limit will prove to be a good idea, once CPU performance is taken into account.

RecTracksDistanceMatrix

Since the computation of the crossing points is costly, a class was introduced that optionally caches those crossing points, as well as the points of closest approach and their distances: the **RecTracksDistanceMatrix**. **LinearizationPointFinders** can be given such a **RecTracksDistanceMatrix**, instead of supplying a simple vector of **RecTracks**.

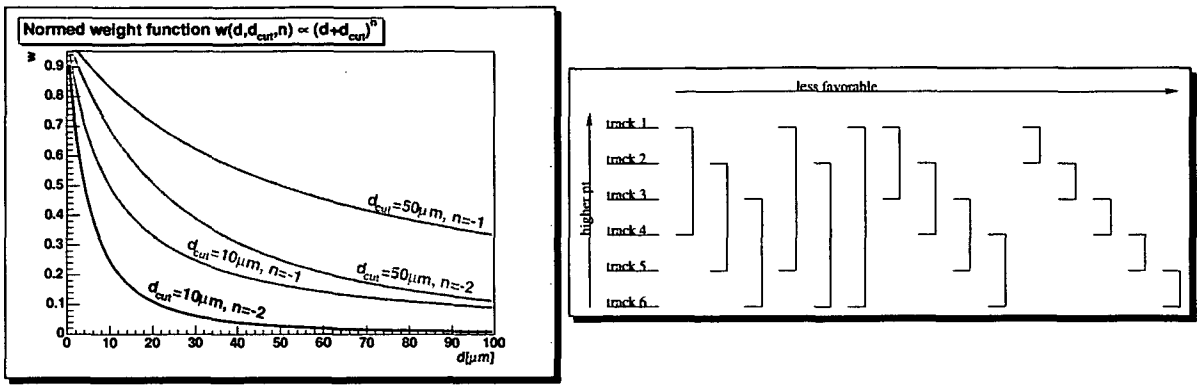


FIGURE 5.2: The weight, as used in the input of the *LinearizationPointFinders*, as a function of the distance. A few typical parameter values are shown (left plot). The sorting mechanism of the crossing point based algorithms shown for the example of 6 tracks (right plot).

5.2.2 Implementations

The following list of crossing-point-based algorithms has been implemented and tested:

- LMSLinearizationPointFinder - coordinate-wise LMS (3.2).
- HSMLinearizationPointFinder - coordinate-wise HSM (3.2).
- SubsetHSMLinearizationPointFinder works just as the above, it only filters out the crossing points with a large "distance" attribute, i.e. crossing points whose associated points of closest approach are very far from one another.
- FSMWLinearizationPointFinder - coordinate-wise FSMW (3.2).
- SMSLinearizationPointFinder - A 3d algorithm (3.4).

For debugging and benchmarking the following finders were implemented:

- ZeroLinearizationPointFinder - always returns (0,0,0).
- MonteCarloBasedLinearizationPointFinder - returns the position of the associated SimVertex.
- LinPtFinderFromVertexFitter Delegates the task of finding a linearization point to a *VertexFitter*.

Finally, an ApexPointBasedLinearizationPointFinder has been implemented that delegates the computation to an ApexFitter.

5.2.3 Performance analysis

The implementations were tested in four different topologies. In $c\bar{c}$ and $q\bar{q}$ topologies all tracks (with no prior track filter) were considered as input; the finder had to produce a viable prototype for a primary vertex; the tracks from the secondary vertices served as contamination of the data. In the $J/\psi\ \phi$ and the τ scenarios the finders had to fit the $J/\psi\ \phi \rightarrow K^+K^-\mu^+\mu^-$, and the $\tau \rightarrow \pi\pi\pi$ decay vertex, respectively. The input data was the RecTracks associated with the four/three relevant TkSimTracks — there was no explicit source of contamination. In case of less than two RecTracks the event was dropped. 2000 events per topology were considered. All distances refer to the x-coordinate only. To "fail" in this context means that the linearization point finder produced a prototype that is more than 3mm displaced from the true vertex position (in the x-coordinate) – clearly the word "failure" should not be taken literally in this context. The time was measured on a 2.8 GHz Intel Celeron; the values are given in milliseconds per event. The algorithms considered were:

- SubsetHSM(-1) SubsetHSM, considering all track pairs (i.e. all crossing points)
- HSM(-1) ...
- LMS(-1) ...
- HSM(100) HSM considering a maximum of 100 track pairs.
- FSMW(exp=-2, f=0.4, c=10, np=-1) FSMW, considering all track pairs, fraction=.4, exponent=-2, $d_{\text{cut}} = 10\mu$m.
- FSMW(exp=-2, f=0.4, c=10, np=200) FSMW, just as above, except for a maximum of 200 track pairs considered.
- ISMS(200) Iterative non-weighted SMS, with a maximum of 200 track pairs.
- TrimmingApex(Unweighted,MTV) An apex point based approach, with a TrimmingApexFitter. The weights are ignored, the MTV algorithm is the apex point finder.

5.2.4 Results

Tables 5.1 and 5.2 show the test results, Tables C.1 and C.2 show a more extensive list of the same test. FSMW with a fraction $f = 0.4$, exponential $n = -2$, a cutoff $d_{\text{cut}} = 10\mu$m, and an upper limit $n_{\text{pairs}} = 200$ seems to be a very sound choice for all topologies. With a 3 milliseconds per event on a 2.8 GHz machine it is reasonably fast, too. The current implementation is not (yet) CPU optimized. The aforementioned "test winner" has become the DefaultLinearizationPointFinder. The complete inheritance tree of the DefaultLinearizationPointFinder is depicted in Fig. 5.3.

LinPtFinder	$c\bar{c}$				$q\bar{q}$			
	RMS [μm]	σ_{Fit} [μm]	fail ‰	t [ms]	RMS [μm]	σ_{Fit} [μm]	fail ‰	t [ms]
SubsetHSM(-1)	38	29	0	3.5	37	25	0	4
HSM(-1)	37	27	0	3.4	32	24	0	3.9
LMS(-1)	196	41	2	3.4	237	44	11	3.9
HSM(100)	54	38	0	0.7	48	32	0	0.8
TrimmingApex(Unweighted,MTV)	147	33	16	51.3	202	37	29	59.5
FSMW(exp=-2, f=0.4, c=10, np=-1)	38	27	0	16.1	34	24	0	22.9
ISMS(200)	33	27	0	8.2	30	26	1	8.1
FSMW(exp=-2, f=0.4, c=10, np=200)	49	33	0	3	43	29	0	3

TABLE 5.1: Resolutions and failure rates of different LinearizationPointFinders for quark pairs. See the text for detailed description.

LinPtFinder	$J/\psi\ \phi \to K^+K^-\mu^+\mu^-$				$\tau^\pm \to \pi^\pm\pi^+\pi^-$			
	RMS [μm]	σ_{Fit} [μm]	fail ‰	t [ms]	RMS [μm]	σ_{Fit} [μm]	fail ‰	t [ms]
SubsetHSM(-1)	744	61	194	0.2	1018	780	379	0.1
HSM(-1)	744	61	194	0.2	1020	780	378	0.1
LMS(-1)	833	139	170	0.2	1035	937	384	0.1
HSM(100)	744	61	194	0.2	1020	780	378	0.1
TrimmingApex(Unweighted,MTV)	781	179	260	0.8	876	553	329	0.7
FSMW(exp=-2, f=0.4, c=10, np=-1)	578	63	140	0.3	967	755	323	0.2
ISMS(200)	736	60	162	0.4	1010	876	382	0.2
FSMW(exp=-2, f=0.4, c=10, np=200)	578	64	140	0.2	967	755	323	0.1

TABLE 5.2: Resolutions and failure rates of different LinearizationPointFinders for low-multiplicity jets. See the text for detailed description.

LinPtFinder	$c\bar{c}$ (jetfilter)		$q\bar{q}$ (jetfilter)		$J/\psi\ \phi$		$\tau \to \pi\pi\pi$	
	vtx	< cut	vtx	< cut	vtx	< cut	vtx	< cut
LMS	1969	1849	1963	1881	1901	1120	1578	250
Fsmw-default	1986	1892	1993	1942	1948	1285	1778	272
Zero	443	381	435	385	407	147	167	23
MonteCarlo	1993	1897	2000	1999	1984	1409	1678	348
LinVtxFit	1955	1840	1935	1854	1926	1305	1648	261

TABLE 5.3: Test of the importance of the linearization point w.r.t. the final (adaptive) vertex fit. See the text for further explanations.

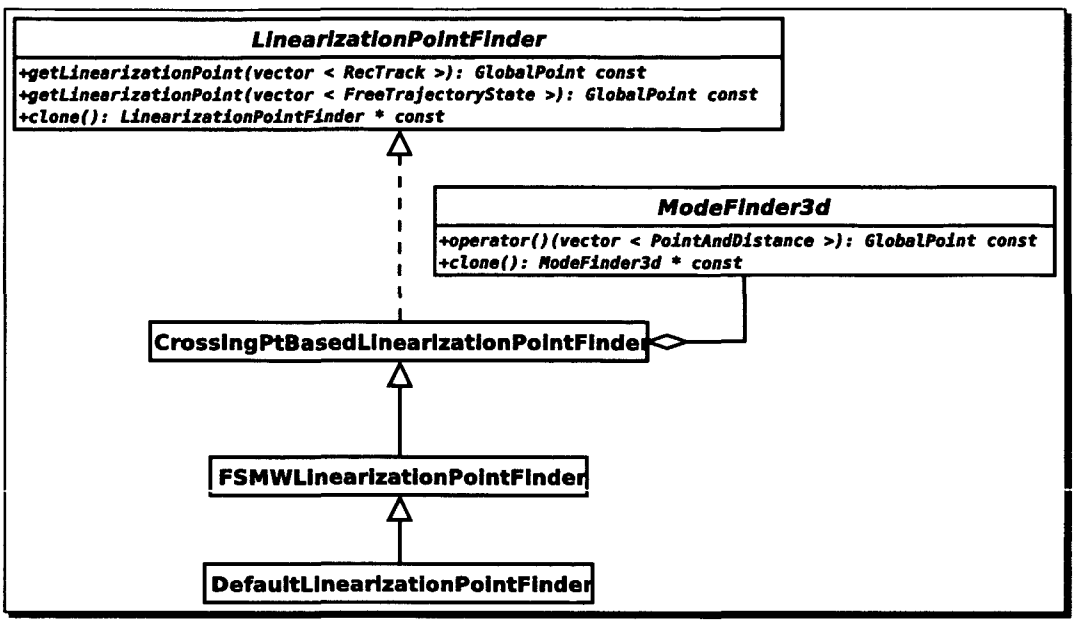


FIGURE 5.3: The DefaultLinearizationPointFinder and its inheritance.

5.2.5 Influence of the linearization point on the final fit

Now that a sensible default linearization point finder has been found, we want to document the influence of the linearization point finder on the final fitted vertex. To this end five different *LinearizationPointFinders* were used as the initial guess of an adaptive vertex fit (see 5.4.2): the LMS algorithm, the "default" FSMW method, the ZeroLinearizationPointFinder, the MonteCarloBasedLinearizationPointFinder, and the LinearVertexFitter. The number of valid RecVertices is compared, as well as the number of RecVertices that are no further than 200μm from the corresponding SimVertex (distance in "full 3d"). For the results consult Table 5.3. The "vtx" column shows the number of RecVertices (see Sec. 5.4.2) that could be associated; association was done by tracks. 2000 events were considered. The "<cut" column indicates how many of the found RecVertices are no further than 200 microns off w.r.t. associated SimVertices. For the final fit the default AdaptiveVertexFitter was employed ($\chi^2_{\text{cut}} = 3.0, T = 64, 9, 4, 1, 1$). The results clearly demonstrate the importance of the linearization point for a robust fitter. The behavior of the LinVtxFit is interesting insofar as it is mathematically equivalent to starting the adaptive method with all weights = constant. The results clearly indicate that such an "interim least-squares step" is not advisable; the chain of robust algorithms should not be broken. Note that in the τ-events the Monte Carlo-based finder is seriously challenged by the FSMW algorithm; the latter actually returns $\approx$ 5% more valid vertices! Fig. 5.4 shows the resolutions plots of the default linearization point finder in two channels.

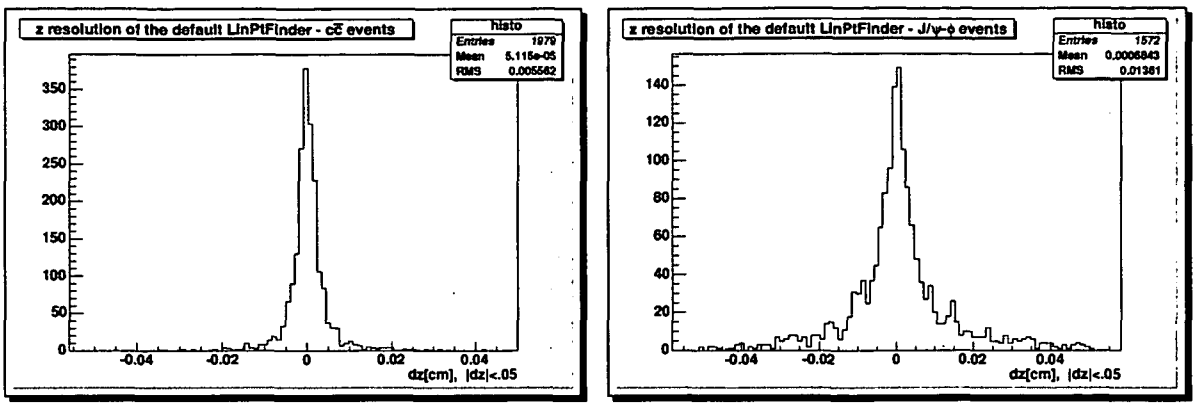


FIGURE 5.4: Resolution plots of the default LinPtFinder.

5.2.6 Conclusions

A solid general purpose linearization point finder has been identified and evaluated. The algorithm has been shown to cover a wide range of applications very well: from a low-multiplicity highly-collimated decay vertex fit to a high-multiplicity primary vertex fit, the default algorithm performs well and is very fast at the same time (Tabs. 5.1, 5.2). The default algorithm with the default settings has been wrapped into a special class, the DefaultLinearizationPointFinder. It is the current default in almost all VertexFitters. The effect of the choice of linearization point finder on a final (adaptive) vertex fit has been documented (Tab. 5.3); it is significant. Whether there is still space for large improvement is difficult to tell. Comparison of the current default algorithm with feeding the Monte Carlo truth (Tab. 5.3) vaguely indicates that there might be some in high multiplicity events. An impressive detail is the performance of the FSMW compared with the Monte Carlo based method — FSMW returns more valid vertices!

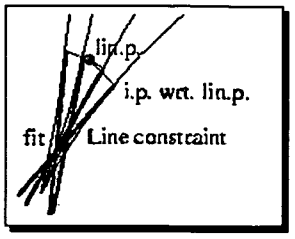
5.3 Least-squares fitting methods

A least squares method *per definition* minimizes the function:

$$\hat{\beta}_{LS} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{n} r_i^2(\beta) \tag{5.2}$$

The derivative of Eq. (5.2) depends linearly on the residual $r_i(\beta)$. In this sense least-squares methods are often called "linear" - every robust method is then non-linear. All vertex fitters also linearize the tracks. In this notation all fitters are linear. If not stated otherwise, the notion of "linearity" will refer to the dependence of the derivative of the objective function on the residuals, so robust fitters will be called non-linear.

In the ORCA framework two different least squares fitting algorithms have been implemented:



- **LinearVertexFitter** This algorithm [70] starts with an initial vertex candidate, then computes the impact points of the tracks with respect to this candidate. The error of the impact point is a 3x3 matrix with a rank of 2. The weighted means of the impact points is now taken; since the error matrix is degenerate, we take the pseudo-inverse, which can be interpreted as having an infinite error in the track direction, which, in turn, describes a track linearized in 6d phase space. If the vertex candidate moves too much within one iteration, the tracks are re-linearized.

- **KalmanVertexFitter** This procedure [51] parametrizes the tracks with a perigee parametrization [27]; it linearizes in the 5 perigee parameters, as opposed to the LinearVertexFitter that linearizes the track in the track's phase space. A standard Kalman filter procedure is now applied to the perigee vectors.

Since both methods minimize the same objective function, they have to converge against the same results, up to numerical instabilities. And indeed the fitters report the same numbers [89]. The perigee parameters are the more sensible choice in the sense that a linearization in these parameters is a closer approximation to reality. The KalmanVertexFitter has been identified as the numerically more stable method.

5.4 Robustifications

The method of Least Squares is seen to be our best course when we have thrown overboard a certain portion of our data – a sort of sacrifice which has often to be made by those who sail the stormy seas of Probability.

F. Y. Edgeworth, 1887

Any kind of robustification can be expressed in terms of changes to the "objective function" of the least squares method (Eq. (5.2)). In this thesis three approaches have been implemented and studied:

- The TrimmingVertexFitter (Sec. 5.4.1) ignores ("trims") a certain fraction of the tracks,
- the AdaptiveVertexFitter (Sec. 5.4.2) down-weights outliers, and
- the LMSVertexFitter (Sec. 5.4.3) minimizes the median — instead of the sum — of the squared residuals.

The three approaches shall now be discussed in greater detail.

5.4.1 The trimmer

The trimming method minimizes the sum of only a subset of all residuals:

$$\hat{\beta}_{LS} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{h<n} r_i^2(\beta) \tag{5.3}$$

The only guaranteed general way to find the global minimum is to try all combinations. Clearly such an exhaustive approach is not feasible in any realistic context. Fortunately we have other methods of guessing the rough whereabouts of our vertex candidate: we have the linearization point! This enables us to implement a very CPU-effective version of P. Rousseeuw's Fast-LTS [46] algorithm. The algorithm can be described as:

Compute a good first guess with e.g.SubsetHSMLinearizationPointFinder.	
	Find the h tracks that are most compatible with the vertex candidate.
	Fit a new vertex candidate with these h tracks
Iterate until convergence	

5.4.2 The adaptive estimator

Let us not throw away data all too hastily. Instead, let us weight and re-weight the data, consider and reconsider alternative models. Only if we must, at the latest possible stage, shall we distinguish between "in" and "out", between signal and noise.

The author, on behalf of the CMS vertexing circle (expressed as a formal answer to Mr. Edgeworth, see p. 77)

The **AdaptiveVertexFitter** [101] does not reject outlying tracks; it rather down-weights them with a well-defined weight w_i:

$$w_i\left(\chi_i^2\right) = \frac{\phi\left(\chi_i^2\right)}{\phi\left(\chi^2_{\text{cutoff}}\right) + \phi\left(\chi_i^2\right)} = \frac{e^{\frac{-\chi_i^2}{2T}}}{e^{\frac{-\chi_i^2}{2T}} + e^{\frac{-\chi^2_{\text{cutoff}}}{2T}}} \tag{5.4}$$

With this definition the objective function now reads:

$$\hat{\beta}_{LS} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{n} w_i \cdot r_i^2(\beta) \tag{5.5}$$

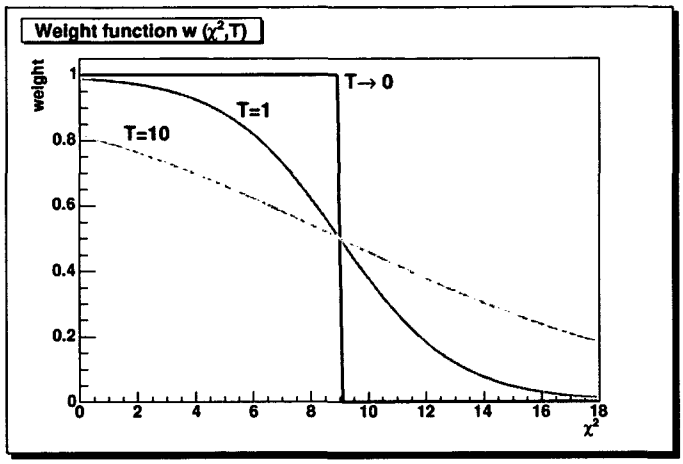


FIGURE 5.5: The weight function (Eq. 5.4).

The fitter is implemented as an iterative re-weighted least squares algorithm; as always, one starts with an initial guess: the linearization point. Tracks weights are computed for the use in a weighted fit. The weights are then re-computed and the vertex re-fitted, until convergence. As an additional improvement a geometric annealing schedule is introduced: with every iteration step the temperature T is multiplied with a factor $r, r \leq 1$.

Note that the distribution function inserted for $\phi(\chi_i^2)$ in Eq. (5.4) does not have to be Gaussian; in principle any distribution could be inserted. If the χ_i^2 follow a χ^2 distribution, the above choice is optimal.

Implementation

The adaptive fitter (AVF) has been implemented in ORCA as a *VertexFitter* (see Fig. 5.6). It uses a *LinearizationPointFinder* (default is DefaultLinearizationPointFinder) for the initial "rough guess" of the vertex. A *VertexUpdator* (default: KalmanVertexUpdator) is then used to sequentially update the vertex candidates. An *AnnealingSchedule* manages the annealing procedure. A *VertexSmoother* is finally called to smooth the vertex. The default smoother is the DummyVertexSmoother. It leaves the vertex untouched. The *AnnealingSchedule* is the only class that has been introduced specifically for the adaptive method, all other classes are reused from the implementations of the linear methods. Note that *polymorphism* has been heavily used in the design of the fitters. All polymorphic classes are cloneable to facilitate object lifetime management.

Small remark on χ^2 computation

When talking about χ^2, we indeed have to be a little more careful. The χ^2 value used in the weight function Eq. (5.4) is a little different from the fitted χ^2. A formal derivation of the adaptive method (Sec. 5.10) makes it evident that

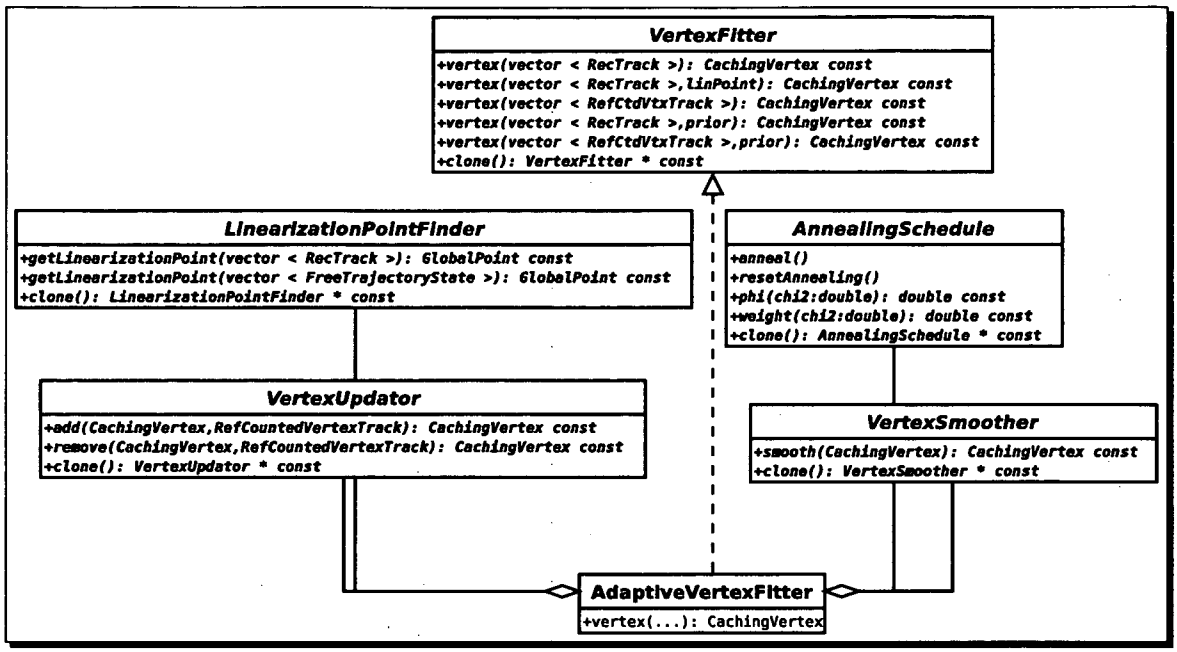


FIGURE 5.6: Implementation of the adaptive vertex fitter

- the weights must not appear in the "weight" χ^2,
- the "vertex term" does not appear, either, in the χ^2 used in Eq. (5.4)

For completeness, here is a list of the three different χ^2 definitions:

$$\chi^2_{k,\text{Linear}} = (\delta\vec{x}_k)\left(\mathbf{C}_{v,k-1} + \frac{\mathbf{C}_{p,k}}{w_k}\right)^{-1}(\delta\vec{x}_k)^T, \tag{5.6}$$

$$\chi^2_{k,\text{Kalman}} = (\delta\vec{x}_k)\,\mathbf{C}^{-1}_{v,k-1}\,(\delta\vec{x}_k)^T + \vec{r}_k w_k \mathbf{C}^{-1}_{p,k}\vec{r}_k^T \tag{5.7}$$

$$\chi^2_{k,\text{weight}} = (\delta\vec{x}_k)\,\mathbf{C}^{-1}_{p,k}\,(\delta\vec{x}_k)^T \tag{5.8}$$

where $\delta\vec{x}_k$ denotes the vertex displacement at iteration k ($\delta\vec{x}_k \equiv \vec{x}_k - \vec{x}_{k-1}$), $\mathbf{C}_{p,k}$ denotes the error matrix of track k, $\mathbf{C}_{v,k}$ the error matrix of the vertex after having added track k. w_k is the assignment probability of track k to the considered vertex and $\vec{r}_k$ denotes the residuals of track k.

Weighted linear fitters

The generalization of a linear (Kalman) method to a more general weighted linear method is relatively straightforward. One has to bear in mind only two issues:

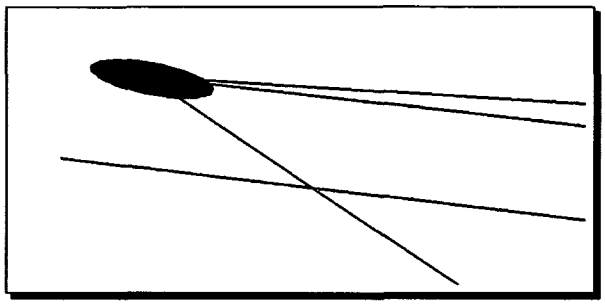


FIGURE 5.7: Result of an adaptive fit. The fitter was supplied with 4 tracks ($K^+K^-\mu^+\mu^-$), one of which is incompatible with the other three. The fitter completely ignores this outlying track.

- The number of degrees of freedom (NDF) is multiplied with the assignment probability; this can of course trigger a software technical change: the representation of the NDF may change from an integral to a floating point type.
- Every occurrence of the covariance matrix of the track parameters has to be divided by the assignment probability.

5.4.3 The Least Median of Squares (LMS)

Generally, the LMS method minimizes the median of the squared residuals, instead of the sum.

$$\hat{\beta}_{LS} = \underset{\beta}{\operatorname{argmin}} \operatorname{med}_{i=1}^{n} r_i^2(\beta) \tag{5.9}$$

Unfortunately the general case has neither a simple analytical solution nor is there a (known) fast numerical algorithm to find that minimum. In ORCA a coordinate-wise LMS that operates on the impact points has been tried. It is a **ModeFinder3d** that works just like the **LMSLinearizationPointFinder**, except that the error is estimated as well. This approach clearly has the disadvantage that one intentionally discards the important information of the spatial structure of the data. And indeed, we have to humbly accept the fact that this algorithm is neither very exact, nor is it very (statistically) efficient.

Error estimation

There is no straightforward well-justified way of computing the error of the LMS method. In order to still deliver an error estimate, all data points are classified into inliers and outliers according to a χ^2-like criterion that is designed to give the "right" χ^2 in the case of Gaussian errors [49]:

$$\tilde{\chi}^2 = 1.4826 \cdot \left(1 + \frac{5}{n-1}\right) \operatorname{med}(x) \left(\operatorname{med}(\mathbf{C}_x)\right)^{-1} \operatorname{med}(x)^T \tag{5.10}$$

Note that the error computation has never been thoroughly tested. Still, it is a robust fitter with a very high breakdown point (50%), and it is faster than the linear vertex fitters.

Implementation

ORCA's **LMSVertexFitter** is implemented as an *Adapter* – a class that bridges the ***ModeFinder3d*** classes with the ***VertexFitter*** interface. The error estimation is currently performed the way described in the previous paragraph. This computation of the error is currently a part of the Adapter. Hence, the user can transform any ***ModeFinder3d*** into a ***VertexFitter***, but the computation of the error estimate is mathematically "justified" only for the LMS algorithm. (Although, since the error estimate is already quite poor for the LMS case, we can assume that it works just as well/poorly with any other mode finder).

5.4.4 Verification of the robust methods

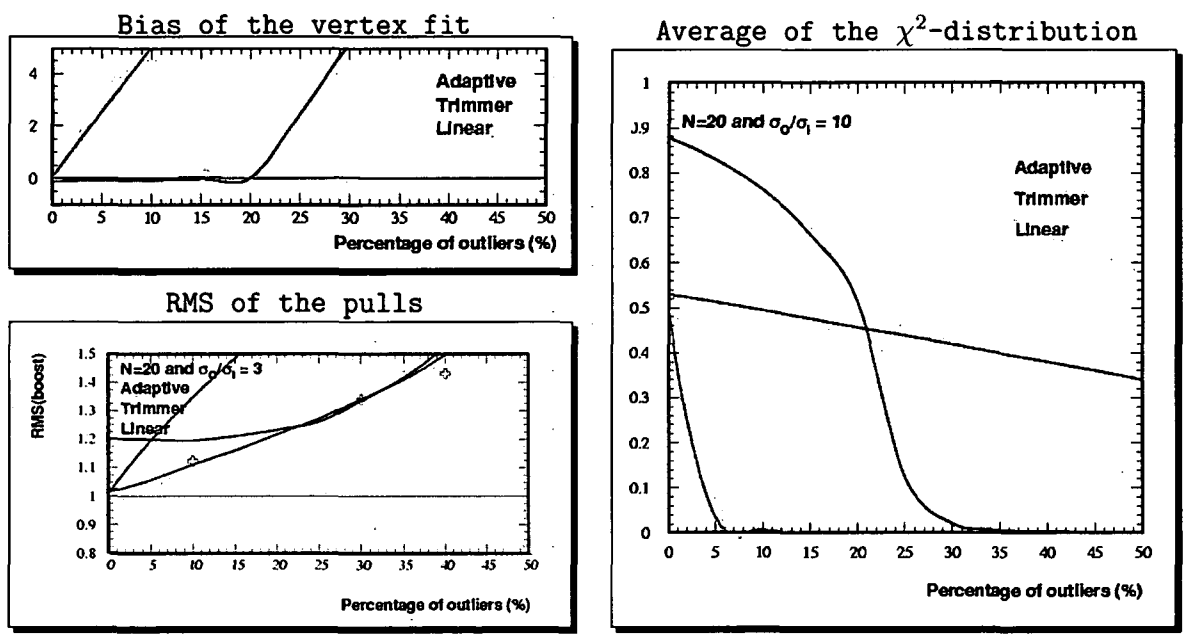


FIGURE 5.8: Plots of different statistical tests, taken from [43]. See the reference for precise definitions of the tests.

The statistical properties of the AVF and the trimming algorithm have been verified and tested extensively in [43]. Fig. 5.8 shows a few tests.

The top left plot shows the bias of the vertex in the presence of "type-2" outliers.[1] The outliers come from another vertex 5mm displaced from the "signal" vertex. The track

[1] Type-2 outliers are tracks that come from the wrong vertex. Type-1 outlier tracks are from the right vertex, but have a mis-measured covariance matrix

multiplicity is always 20, including outliers.

The bottom left plot shows the RMS of pulls in the presence of type-1 outliers.

The right plot shows the mean of the χ^2 distribution in the presence of "type-1" outliers. The outliers are unbiased but their true errors are 10 times their covariance matrix. Clearly visible are the linear fitter's "ideal" 0.5 at the presence of no outliers, then the rapid decline. The trimmer exhibits very similar properties only 20 % "further right" (the trimming fraction is 20%). The adaptive method has a typical slight error over-estimation at 0% outliers, then remains fairly neutral to the additional noise.

5.4.5 Conclusions

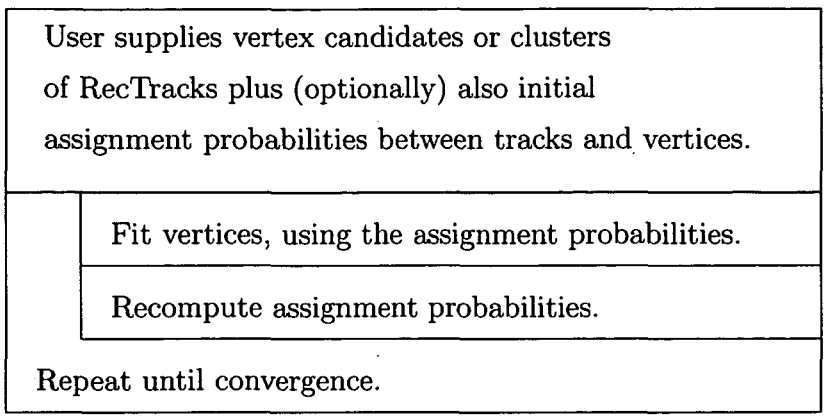
From these tests with artificial data alone it is already very obvious that the **AdaptiveVertexFitter** is a very robust, general purpose algorithm. It can operate with very pure data with almost no loss in statistical efficiency, while it remains quite unimpressed by the presence of contamination. As opposed to many other robustification schemata no special assumptions about the data have to be supplied. The trimmer exhibits the same behavior as the linear fitter, only "shifted" to the right by the trimming fraction. Specific knowledge of the expected outlier fraction would have to be known in order for the trimmer to perform well. Thus, it seems impossible for the trimmer to compete with the much more generic adaptive estimator. Surely, the linear method is still the only efficient method in case of perfect data.

5.5 MultiVertexFit

The **MultiVertexFitter** (MVF) [56] is an algorithm that fits n vertices at once; it is very similar to the **AdaptiveVertexFitter** – it, too, has a soft assignment with a weight function that is indeed a generalization of eq. (5.4); for the special case of one vertex the **MultiVertexFitter** and the **AdaptiveVertexFitter** are equivalent. What changes with $n > 1$ is that the weight function generalizes to:

$$w_{ij} = \frac{e^{\frac{-\chi^2_{ij}}{2T}}}{e^{\frac{-\chi^2_{\text{cutoff}}}{2T}} + \sum_{k=1}^{n} e^{\frac{-\chi^2_{ik}}{2T}}} \tag{5.11}$$

Here w_{ij} denotes the weight of track i with respect to vertex candidate j. Likewise, χ^2_{ij} is the χ^2 compatibility between track i and vertex candidate j. The fitting procedure can now be described as:



Note that technically the MVF is not a *VertexFitter*, since a *VertexFitter* is an algorithm that delivers one and only one CachingVertex. The MVF is its own category; it does not derive from any super-class.

Caching in MVF To speed up the somewhat slow MVF it has been tried to introduce caching of the LinearizedTrackStates. To this end, the standard SequentialVertexFitter had to be adopted to exploit the MVF's cache. So far studies have shown no significant improvements in CPU performance.

5.5.1 Implementation

The MultiVertexFitter is *not* a *VertexFitter*, as can be seen in Fig. 5.9. Similar to the implementation of the AVF it, too, recycles many classes that have been written for the linear vertex fitters. In the case of the MVF, a complete KVF is used internally — the MVF is quite literally implemented as a set of weighted Kalman filters run in parallel. The linearized track states are saved in LinTrackCache between the iteration steps. The *AnnealingSchedule* has been introduced in the AVF and is recycled here.

5.5.2 Verification of the MVF

Events were generated with two vertices. A primary vertex at (0,0,0) had 5 tracks attached to it, the tracks forming a jet with total jet momentum $\vec{p}_{\text{prim}} = (0, 25, 25)$, and opening angle=0.5. At (0,0,0.2) a secondary vertex was positioned with 3 tracks, jet momentum $\vec{p}_{\text{sec}} = (-15, 0, 20)$, opening angle=0.5. All tracks were "perfect", i.e. the errors were Gaussian and correctly described by the tracks' covariance matrices.

Afterwards the track of the primary vertex that is most compatible to the secondary vertex was selected, and vice versa. Those two tracks were moved to the "wrong" vertices — the primary track was assigned to the secondary vertex and vice versa, see Fig. 5.10.

These two track bundles with one mis-associated track in each bundle were now the input for the verification procedure that compares the MVF to the AVF and the KVF.

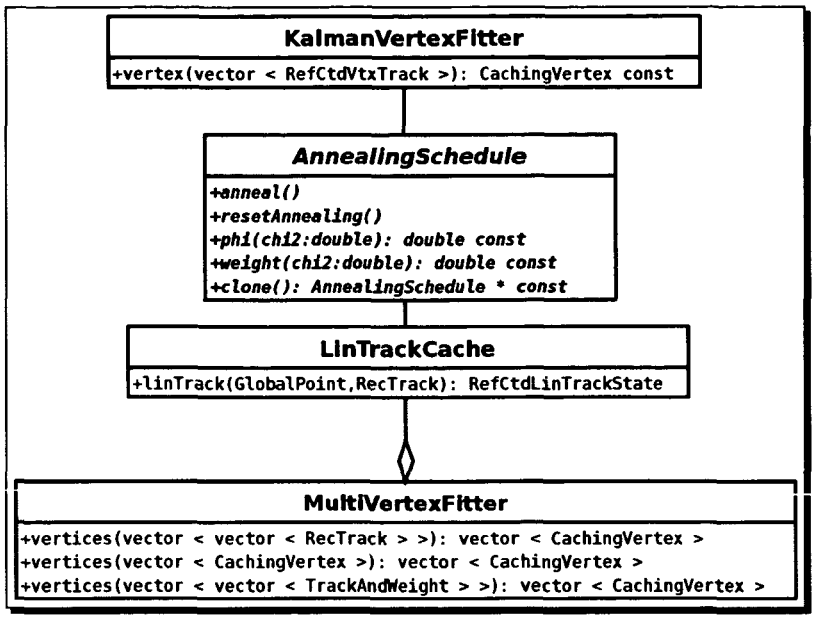


FIGURE 5.9: Implementation of the multi vertex fitter

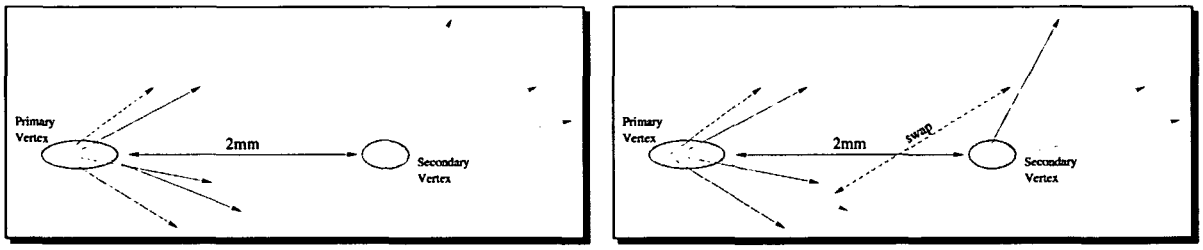


FIGURE 5.10: MVF code verification: Track swapping

Fig. 5.11 shows a comparison of the final flightpath resolutions[2], and Fig. 5.12 shows the assignment probabilities of each track to each vertex of the AVF versus the MVF. Note that the AVF and the KVF vertices cannot recover the mis-associated tracks. The AVF can down-weight the outliers, but not recover the tracks that landed in the wrong track bundle. The effect of this missing information on the "flightpath resolution" is nicely documented in Fig. 5.11; the KVF has a huge bias, the AVF can down-weight the outlying tracks that cause the bias, but it "sees" fewer tracks than the MVF. As a result, the AVF exhibits a wider resolution plot compared to the MVF.

The most direct code verification is Fig. 5.12. Here we see directly the track weights of AVF versus MVF. The plot shows 10000 events with 8 tracks and 2 vertices per track, resulting in 80000 correct track-to-vertex associations ("inliers") and 80000 wrong associa-

[2] The flightpath is the distance between the primary vertex and the secondary vertex.

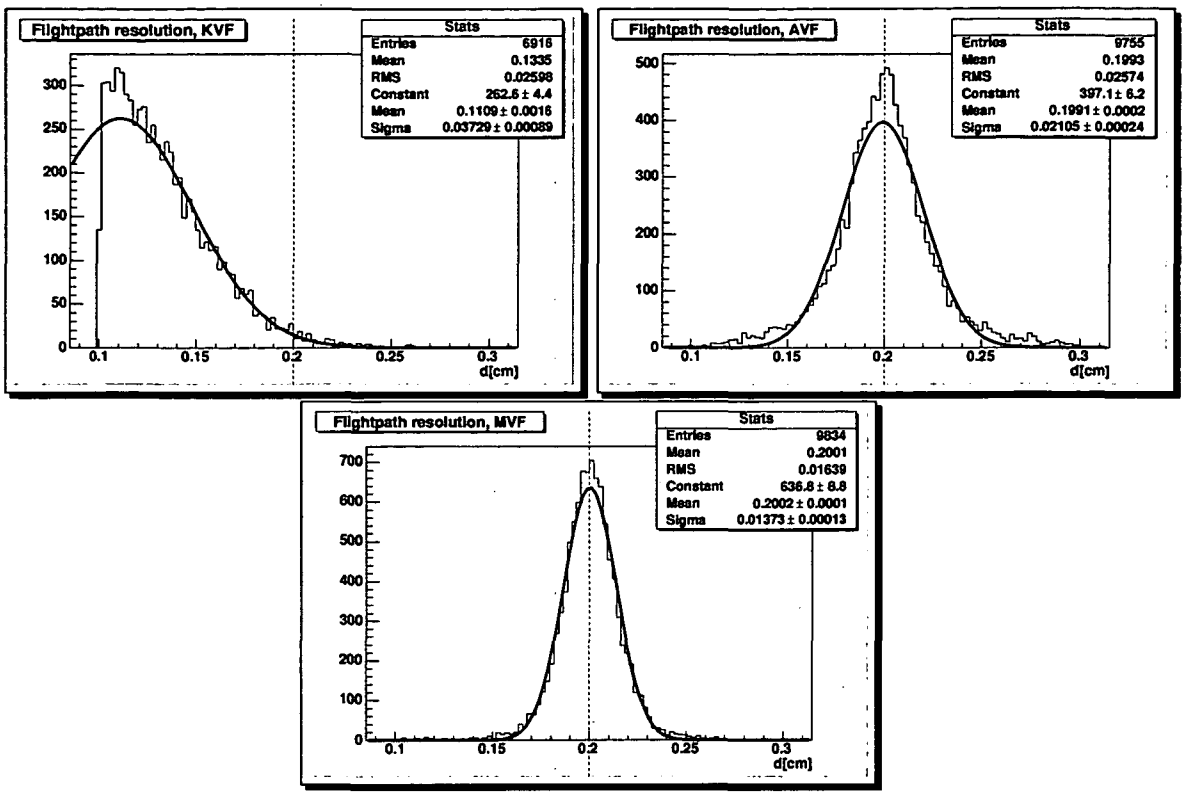


FIGURE 5.11: Flightpath resolutions for the different fitters. The blue line denotes the true value.

tions ("outliers"). It can be seen that both algorithms almost always identify the outliers. The AVF fails to identify only three outliers; the MVF is a bit worse in this case: roughly 30 tracks are assigned to a vertex that is not the right one. In the case of inliers an ideal AVF can only do right in 60000 cases; 20000 tracks are mis-associated from the very beginning and cannot be recovered. We see that the implementation comes very close to the theoretical limit. Only a rough 450 inlying tracks (less than one percent) are down-weighted when they should not be ($\chi^2_{\text{cut}} = 16$). The MVF performs similarly with respect to its own theoretical limit, only its theoretical limit is much higher. About 400 tracks have a weight close to zero where the right weight would be one or close to one.

5.5.3 Future ideas for the MVF

A short term development of the MVF will be the introduction of additional "hard assigned" tracks; tracks that are associated to one specific vertex, with the association being in a "frozen" state. This additional feature will be used to exploit knowledge that comes from external sources. As a longer term development a generalization of these "hard assigned" tracks would be desirable; it would be nice to implement a framework for any kind

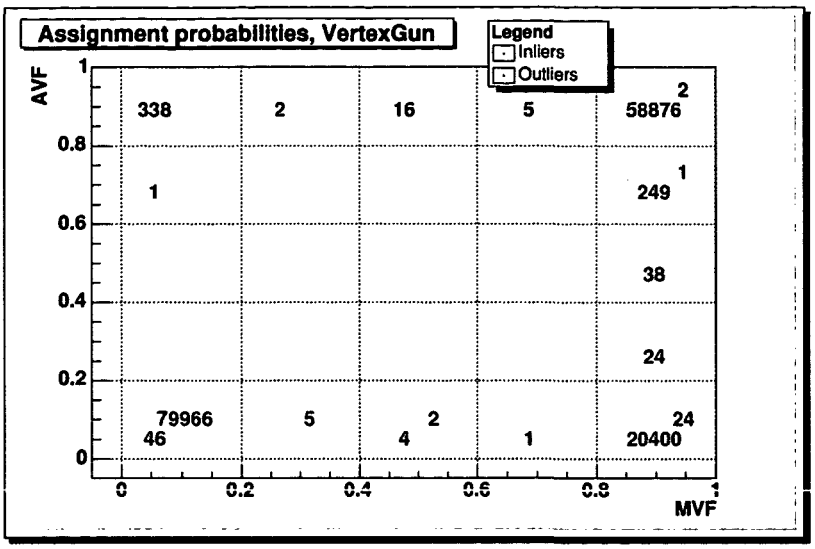


FIGURE 5.12: Comparison of the assignment probabilities, AVF versus MVF.

of constraints, be it on the tracks, its weights, or a quantity related to the vertex. A way to achieve this could be to interact with the kinematic package [79]. This package implements kinematic constraints via the Lagrange multiplier formalism.

5.6 Prior information

A vertex fit can also make use of a prior knowledge of the vertex. This prior information is used as a linearization point with finite errors. The number of degrees of freedom is raised by three. All our fitter implementations can deal with such a prior information. One use case for this feature is to feed a fitter with the knowledge of the beam profile. This makes sense if it is known that the vertex that is to be fitted is a primary vertex.

5.7 The Gaussian Sum Filter (GSF)

The GsfVertexFitter (GSF) [53] is the vertex fit's equivalent to the GsfTrajectoryFitter [20, 21]. It is not part of this PhD thesis. It appears here merely because it is possible to combine the GSF and the AVF - see next section. In the vertex fitting context the concept of the GSF may be described as follows: the track parameters in ORCA can not only be described in a Gaussian model, but also as a mixture — a sum — of a number of Gaussians. Each component of the mixture is assigned a weight. This more detailed information can be exploited when fitting a vertex. In the Kalman formalism this implies that whenever a vertex is updated with a multi-component trajectory state, the result is a multi-component vertex, the number of vertex components being the previous number of

vertex components times the number of track components. The Bayes theorem is used to compute the component weights. Since the procedure explodes computationally, trimming needs to be performed to keep the number of vertex components at a reasonable number.

5.8 GSF and AVF

The GSF can also be combined with the AVF; one can use the GaussianSumUpdator instead of the default KalmanVertexUpdator in the AdaptiveVertexFitter. That way we end up with a vertex fitter that "decides" for the most plausible component within a TrajectoryState, while it can also down-weight entire TrajectoryStates within a vertex fit. This combined algorithm has not been anticipated; the idea only came after all the necessary code has already been written (except for a very few lines). This fact can be taken as a good confirmation of our design choices.

Fig. 5.13 depicts a few code verification tests. The test shows resolution in x (top row) and the pulls in x (bottom row) of the GSF algorithm (left column) and the combined AdaptiveGSF (right column). Input data were 4 tracks with two Gaussians as the track parameter distribution. The track errors are described correctly, only one track comes from another vertex. Implementation and study of the AdaptiveGSF (including the plots above) are work done by Thomas Speer [53]. No contribution to the AdaptiveGSF has been done in the context of this PhD thesis.

5.9 GSF and MVF

It is conceivable that the MVF is used in conjunction with the GSF: one can use a GSFVertexUpdator in the MVF. This has not been tried yet.

5.10 Derivation of the adaptive methods

Literature so far knows of two ways to motivate the weight function Eq. (5.4) from more general principles. The two deductions shall briefly be performed here; a less powerful but novel derivation from a quantum mechanical analogy will be given afterwards.

5.10.1 Statistical approach

The weights in Eqs.(5.4) and (5.11) have so far been introduced without any further explanation. This section is dedicated to an a posteriori derivation of the algorithms. The adaptive fitter can be seen as a special case of the multi vertex fitter, so we shall from the very start allow for n_v vertices. In that case, the estimation of the vertex can – in full analogy to track fitting [54] – be formulated as the problem of finding the minimum of the

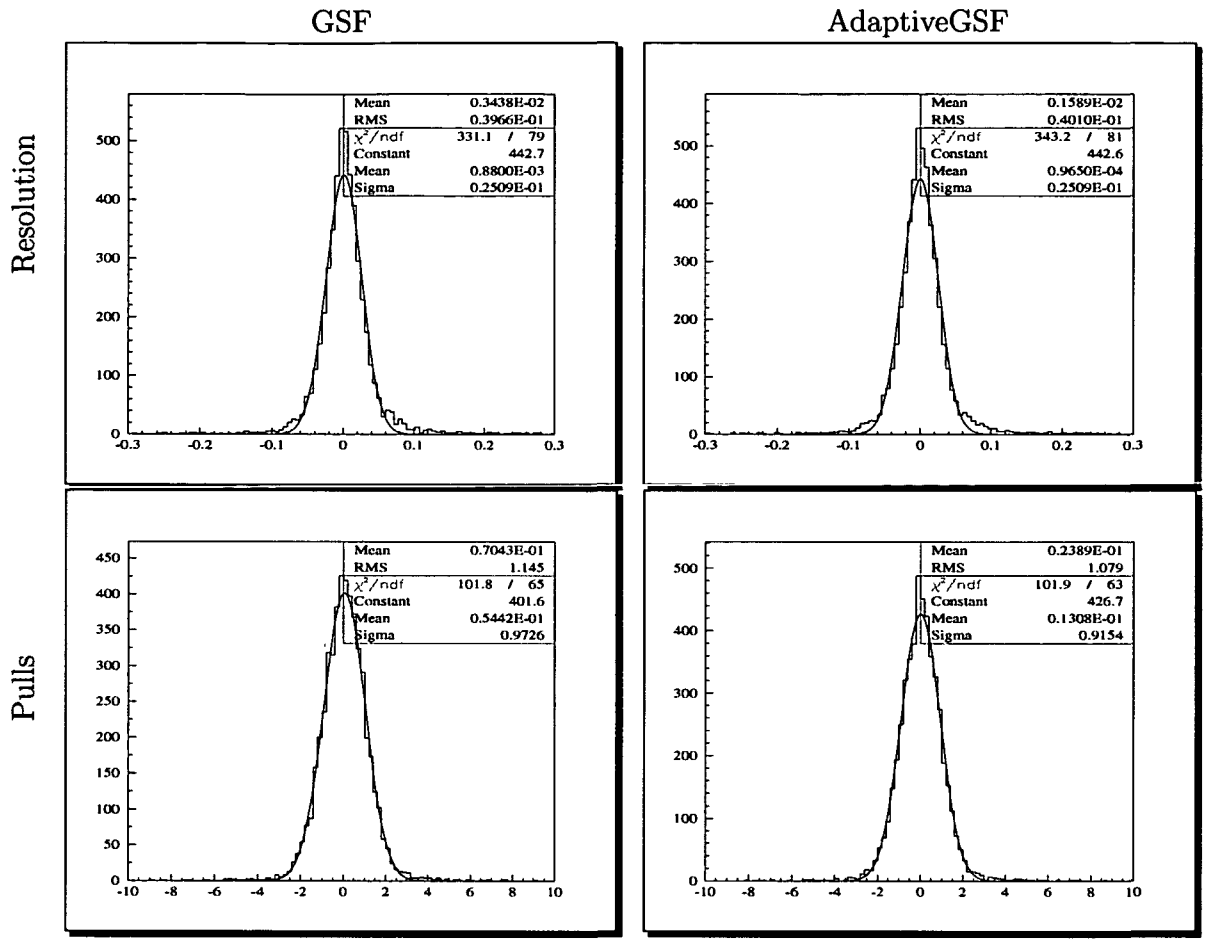


FIGURE 5.13: Comparison of the GSF with the AdaptiveGSF fitter

following objective function:

$$E\left(\{S_{ik}\}, \vec{x_1} \ldots \vec{x_{n_v}}\right) = \sum_{i=1}^{n_t} \left[\sum_{k=1}^{n_v} S_{ik} M_{ik} + \lambda \left(\sum_{k=1}^{n_v} S_{ik} - 1 \right)^2 \right] \tag{5.12}$$

Here x_v denotes the location of the vertex v, the index k runs over the vertices, whereas index i refers to track i. M_{ik} is the squared weighted distance of track i to vertex k. The assignment variables S_{ik} are either 1 or 0; S_{ik} describes whether or not track i is part of vertex k. In the multi vertex fit scenario we introduce competition between the different vertices — again, in full analogy to the DAF track fitting method. Thus we introduce the constraint

$$\sigma_S(i) \equiv \sum_{k=1}^{n_v} S_{ik} = 0 \text{ or } 1. \tag{5.13}$$

Usually it takes at least two tracks to fit a vertex. This, too, could in principle be translated into a constraint. It is not clear to the author whether it makes sense to incorporate such an additional constraint. The only thing that seems evident is that it complicates the derivation significantly.

5.10.2 Effective energy

The energy function Eq. (5.12), together with the constraint Eq. (5.13) completely defines the statistical problem. In order to transform the combinatorial problem into a continuous problem we add noise in the form of a Boltzmann distribution [75]:

$$P\left(\{S_{ik}\}|\vec{x}_l\right) = \frac{e^{-\beta E(\{S_{ik}\})}}{Z}, \tag{5.14}$$

where Z is the partition function:

$$Z = \sum_{\{S_{ik}\}} \int d^{3n_v}\vec{x}_l e^{-\beta E(\{S_{ik}\})} \tag{5.15}$$

β denotes the inverse temperature $1/T$. The next vital ingredient is the marginal probability

$$P_M\left(\vec{x}_l\right) = \sum_{\{S_{ik}\}} P\left(\{S_{ik}\}|\vec{x}_l\right) \tag{5.16}$$

Not all configurations of $\{S_{ik}\}$ satisfy the constraint Eq. (5.13). Let the "legal" set of configurations $\mathcal{S}$ be defined as:

$$\begin{aligned} \mathcal{S} &= \{S|S \text{ legal}\} \\ S \text{ legal} &\Leftrightarrow \sigma_S(i) = 0 \text{ or } 1 \ \forall i \end{aligned} \tag{5.17}$$

The marginal probability is now computed:

$$\begin{aligned} ZP_M\left(\vec{x}_l\right) &= \sum_{S\in\mathcal{S}} e^{-\beta E} \\ &= \sum_{S\in\mathcal{S}} e^{-\beta \sum_{i=1}^{n_t}\left[\sum_{k=1}^{n_v} S_{ik}M_{ik} + \lambda\left(\sum_{k=1}^{n_v} S_{ik} - 1\right)^2\right]} \end{aligned} \tag{5.18}$$

We introduce the function

$$\alpha_S(i) = \begin{cases} 0 & \text{if } \sigma_S(i) = 0 \\ \text{vertex index of track } i & \text{if } \sigma_S(i) = 1 \end{cases} \tag{5.19}$$

It is thus always true that:

$$0 \leq \alpha_S(i) \leq n_v \tag{5.20}$$

With the αs the marginal probability function can be rewritten:

$$\begin{aligned} ZP_M(\vec{x}_l) &= \sum_{S\in\mathcal{S}} e^{-\beta\sum_{i=1}^{n_t}\left[\delta_{\sigma_S(i),1}M_{i\alpha_S(i)}+\delta_{\sigma_S(i),0}\lambda\right]} \\ &= \sum_{S\in\mathcal{S}}\prod_{i=1}^{n_t} e^{-\beta\left[\delta_{\sigma_S(i),1}M_{i\alpha_S(i)}+\delta_{\sigma_S(i),0}\lambda\right]} \\ &= \sum_{S\in\mathcal{S}}\prod_{i=1}^{n_t} e^{-\beta\left[\left(1-\delta_{\alpha_S(i),0}\right)M_{i\alpha_S(i)}+\delta_{\alpha_S(i),0}\lambda\right]} \\ &= \sum_{S\in\mathcal{S}}\prod_{i=1}^{n_t}\left[\delta_{\alpha_S(i),0}e^{-\beta\lambda}+\left(1-\delta_{\alpha_S(i),0}\right)e^{-\beta M_{i\alpha_S(i)}}\right] \end{aligned} \tag{5.21}$$

Summing up over the legal configuration yields:

$$ZP_M(\vec{x}_l) = \prod_{i=1}^{n_t}\left[e^{-\beta\lambda}+\sum_{k=1}^{n_v}e^{-\beta M_{ik}}\right] \tag{5.22}$$

The effective energy can now be defined as:

$$E_{\text{eff}} = -\frac{1}{\beta}\ln(ZP_M) = -\frac{1}{\beta}\sum_{i}^{n_t}\ln\left(e^{-\beta\lambda}+\sum_{k}^{n_v}e^{-\beta M_{ik}}\right). \tag{5.23}$$

Local extrema fulfill:

$$\frac{\partial E_{\text{eff}}}{\partial\vec{x}_l} = \sum_{i}^{n_t}\left(\frac{e^{-\beta M_{il}}}{e^{-\beta\lambda}+\sum_k^{n_\nu}e^{-\beta M_{ik}}}\frac{\partial M_{il}}{\partial\vec{x}_l}\right) = 0 \tag{5.24}$$

$$\sum_{i}^{n_t}\left(w_{il}\frac{\partial M_{il}}{\partial\vec{x}_l}\right) = 0 \tag{5.25}$$

with the weights w_{il} that are exactly as in Eq. (5.11).

5.10.3 EM algorithm

Another theoretical motivation for the weights Eq. (5.11) can be given via the well known expectation-maximization (EM) algorithm [42]. The EM algorithm can be seen as an extension of the maximum-likelihood method in the presence of non-observable data. We consider the tracks $\vec{p}_i$ to be our observed data; the vertex positions $\vec{x}_k$ are the parameters that are to be estimated. y_i denotes the vertex index of track i ($0 \le y_i \le m, i = 1\cdots n$) and is considered unobservable. Assigning a value of zero to the index y_i denotes a track that is not assignable to any vertex. To make for a more pleasurable reading experience we shall also introduce $\vec{\theta} = (\vec{x}_1, \cdots, \vec{x}_m)$. The complete density function now reads:

$$f_c\left(\vec{p}_1,\cdots,\vec{p}_n,y_1,\cdots,y_n|\vec{\theta}\right) = \prod_{i=1}^{n} f\left(\vec{p}_i,y_i|\vec{\theta}\right) \tag{5.26}$$

The partial density function $\left(\vec{p}_i, y_i|\vec{\theta}\right)$ is in our case:

$$f\left(\vec{p}_i, y_i|\vec{\theta}\right) = \left(\vec{p}_i|y_i = k, \vec{\theta}\right) p\left(y_i = k|\vec{\theta}\right) = e^{-\frac{1}{2T}M_{ik}} p\left(y_i = k|\vec{\theta}\right) \tag{5.27}$$

As always, zero correlations between the tracks were assumed. The prior probabilities $p\left(y_i = k|\vec{\theta}\right)$ can be used to specify prior knowledge of track-to-vertex association. If the prior is non-informative (uniform), as is usually the case, it drops out of the formalism. For the rest of the derivation, such a non-informative prior will be assumed. According to the maximum-likelihood principle, the estimate of $\vec{\theta}$ is the value that maximizes the log-likelihood of the observed data:

$$l_0\left(\vec{\theta}; \vec{p}_1, \cdots, \vec{p}_n\right) = \ln f_0\left(\vec{p}_1, \cdots, \vec{p}_n|\vec{\theta}\right) = \ln \sum_i f_c\left(\vec{p}_1, \cdots, \vec{p}_n, y_1, \cdots, y_n|\vec{\theta}\right) \tag{5.28}$$

Further,

$$f\left(y_i = k|\vec{p}_i, \vec{\theta}\right) = \frac{f\left(y_i = k, \vec{p}_i|\vec{\theta}\right)}{f\left(\vec{p}_i|\vec{\theta}\right)} = \frac{e^{-\frac{1}{2T}M_{ik}\left(\vec{\theta^{(t)}}\right)}}{\sum_{k=0}^{m} e^{-\frac{1}{2T}M_{ik}\left(\vec{\theta^{(t)}}\right)}} \tag{5.29}$$

Since $y_i = 0$ denotes non-assignable tracks, M_{i0} introduces a "cutoff" for true outliers: $M_{i0} = M_{\text{cut}}$.

The EM algorithm consists of two steps:

- The E (expectation) step

$$\begin{aligned} Q\left(\vec{\theta}|\vec{\theta}^{(t)}\right) &= \mathbb{E}\left(l_c(\vec{\theta}; \vec{p}_1, \cdots, \vec{p}_n, y_1, \cdots, y_n)|\vec{p}_1, \cdots, \vec{p}_n, \vec{\theta}^{(t)}\right) \\ &= \sum_i \ln f_c\left(\vec{p}_i, y_i|\vec{\theta}\right) f\left(y_i|\vec{p}_i, \vec{\theta}^{(t)}\right) \end{aligned}$$

  translates into the computation of the weights, while

- the M (maximization) step

$$\vec{\theta}^{(t+1)} = \text{argmax}_{\vec{\theta}} Q\left(\vec{\theta}|\vec{\theta}^{(t)}\right)$$

  re-fits the vertices given the computed weights.

For the computation of the weights:

$$Q\left(\vec{\theta}|\vec{\theta}^{(t)}\right) = \sum_{i=1}^{n} -\frac{1}{2T} M_{ik} \sum_{k=1}^{m} \frac{e^{-\frac{1}{2T}M_{ik}}}{\sum_{l=1}^{m}} e^{-\frac{1}{2T}M_{il}} + e^{-\frac{1}{2T}M_{\text{cut}}} = \sum_{k=1}^{m}\sum_{i=1}^{n} -\frac{1}{2T} M_{ik} w_{ik} \tag{5.30}$$

Again, the choice of the weight function in Eq. (5.11) has been derived from basic principles.

5.10.4 Quantum mechanics

Another interesting proposal for a pattern recognition algorithm has been proposed recently [64]. The method exploits an analogy with quantum mechanics: the data points are "smeared out" with a Gaussian distribution, then superimposed. This distribution is interpreted as a quantum mechanical wave function. The minima of the corresponding potential are then considered the center of the clusters. This section shall establish a relationship between this quantum mechanical metaphor and the adaptive methods.

Let $\vec{x}_i$ denote the spatial position of a data point i, and $\mathbf{V_i}$ denote its error. One can now define:

$$\delta\vec{x}_i \equiv (\vec{x} - \vec{x}_i) \tag{5.31}$$

$$\chi_i^2 \equiv \chi^2(\vec{x};\vec{x}_i) \equiv \delta\vec{x}_i^T \mathbf{V_i}^{-1} \delta\vec{x}_i \tag{5.32}$$

$$\phi_i(\vec{x}) \equiv \frac{1}{(2\pi)^{\frac{3}{2}}\sqrt{T|\mathbf{V_i}|}} e^{\frac{-\chi_i^2}{2T}} \tag{5.33}$$

In the above equation a (scalar and dimensionless) "temperature" T was introduced. The normed sum of the ϕ_i

$$\psi(\vec{x}) = \frac{1}{n}\sum_i \phi_i(\vec{x}) \tag{5.34}$$

shall be interpreted as the lowest ground state of the Schrödinger equation

$$H\psi(\vec{x}) = \left(-\frac{\hbar^2}{2m}\vec{\nabla}^2 + V(\vec{x})\right)\psi(\vec{x}) = E\psi(\vec{x}) \tag{5.35}$$

with

$$\vec{\nabla}^2\psi = \frac{1}{n}\sum_i \left[\frac{\delta\vec{x}_i^T \mathbf{V_i}^{-1}\mathbf{V_i}^{-1}\delta\vec{x}_i}{T^2} - \frac{1}{T\,\mathrm{tr}(\mathbf{V_i})}\right]\phi_i \tag{5.36}$$

At this point unfortunately generality has to be sacrificed. We specify to the unpleasantly special case that the error matrices are the same and a multiple of the unit matrix:

$$\mathbf{V_i} \equiv V \cdot \mathbb{1} \tag{5.37}$$

This constraint reduces the covariance matrix to a simple scale factor that will eventually be expressed as a function of T. With this constraint we can formulate:

$$\vec{\nabla}^2\psi = \frac{1}{n}\sum_i \frac{\chi_i^2}{T^2V}\phi_i - \frac{1}{3TV}\psi \tag{5.38}$$

Since V is degraded to an overall scale factor, it serves a similar purpose as the temperature T that we introduced at the beginning. T can thus be described as a function of V. We choose:

$$T = \sqrt{\frac{\hbar^2}{2nmV}}$$

. Choosing the total energy of our system to be $E = \frac{n}{3}T$, one arrives at

$$V(\vec{x}) = \sum_i \chi_i^2 \frac{\phi_i}{\sum_j \phi_j} \tag{5.39}$$

The task of minimizing this function is now again mathematically fully equivalent to introducing the weights in Eq. (5.4), except for the lack of the χ^2_{cutoff} term. Unfortunately the covariance matrices of the data points in Eq. (5.37) had to be discarded. This breaks the analogy. Still, a striking similarity is visible between the adaptive method and this method which introduces a clustering algorithm via a quantum mechanical analogy. Possibly, a (classical) field theory could be used as well.

CHAPTER 6

Performance studies

This chapter is dedicated to in-depth studies of vertex reconstruction strategies in selected, relevant and realistic physics channels. Both vertex fitting and vertex finding algorithms are covered. Special emphasis is put on application of vertex reconstruction strategies to the b-tagging task.

6.1 Vertex fitting

The PrincipalVertexReconstructor (see Sec. 4.4.2) can, if we consider only the "leading vertex", be seen as a robust vertex fitter that works with hard assignment and is based on the statistical track-to-vertex compatibility. Considering this, it is of interest to compare the PVR with the AdaptiveVertexFitter; it is a "fair" comparison between a method that works with soft assignment and a similar one based on hard assignment.

This comparison was performed in four channels: $c\bar{c}$ (jet filtered), $q\bar{q}$ (jet filtered), $J/\psi\ \phi$ and $\tau \rightarrow \pi\pi\pi$. In the first two cases the primary vertex had to be fitted, with the tracks from the hadronic jets serving as a source of contamination of the data. In the latter two channels no mis-associated tracks were considered. Mis-measured tracks were the only source of contamination. The standard CombinatorialKalmanTrackFinder was used for track reconstruction. A lot of attention was turned to making sure that the comparison is fair: both algorithms use the DefaultLinearizationPointFinder and the KalmanVertexUpdator. For the PVR a *facade* class was written: the PrincipalVertexFitter (PVF): a vertex fitter that returns the vertex with the highest track multiplicity found by the PrincipalVertexReconstructor. This PrincipalVertexFitter used the VertexCompatibility3D to determine track-to-vertex compatibility. In order to have a legitimate baseline, both algorithms were also compared with the standard KalmanVertexFitter. The study also tried to evaluate how "tunable" these algorithms are, how much the result depends on the most sensitive parameter. For the AdaptiveVertexFitter this parameter clearly is the "cutoff" parameter. The annealing schedule has already been shown to play a less crucial role; the default ($T_{ini} = 256; r = 0.25$) schedule was considered. In the PVF case it was decided to change both the `VertexTrackCompatibility` and the `VertexFitCompatibility` criterion *in sync* - one parameter had always the same value as the other. The fitters were run over 5000–10000 events per channel.

6.1.1 Results

Figs C.1, C.2, C.3, C.4 in the appendix nicely demonstrate the superiority of the AVF over the KVF even in the absence of truly mis-associated tracks (in the $J/\psi\ \phi$ and τ channels). The difference becomes particularly convincing in the presence of outliers, as is the case in the $c\bar{c}$ and the $q\bar{q}$ channels.

A comparison with the PVR is more subtle: Figs 6.1, 6.2 and C.5, C.6 show detailed results of the analysis in various event channels. A few general tendencies can be summarized as follows: Both algorithms produce roughly the same results. The resolution plots as well as the plots of the standardized residuals look very much alike. One clear asset of the adaptive method is that it is more reliable: in all event channels there has not been one single "failure". When plotting the resolutions or the standardized residuals, the AVF tends to produce more events in the same interval. Sometimes this difference in the number of events in a given interval can be quite significant (e.g. 5464 events versus 4744 in the z resolution plot in the $J/\psi\ \phi$ channel, see Fig. 6.1). Only the z resolution plots in the τ channel deviate slightly from this rule. Another major difference is the CPU consumption,

as the following table shows:

	CPU time per event	
	PVF	Adaptive
$J/\psi\ \phi$	3 ms	3 ms
τ	2-3 ms	2-3 ms
$c\bar{c}$	100 ms	24 ms
$q\bar{q}$	120 ms	25 ms

It is, though, important to note that the PVF in its current implementation still fits more than one vertex. The secondary vertices are discarded only afterwards. The table given above will have to be re-evaluated once a stopping mechanism is introduced in the PVR/PVF.

Ultimately it is safe to state that the AVF should be treated as the new default method. In realistic scenarios it outperforms least-squares techniques, even in the absence of truly mis-associated tracks. The PVF and the AVF perform roughly equally well, but the AVF tends to be more reliable and a lot more "efficient" in the sense that in more cases the AVF produces sensible results. The AVF is also the more general approach. The weights that were introduced carry information that can be used in "higher level" code, like a kinematic fit or a b-tagging algorithm. It will be very interesting to see how well this new piece of information can be exploited by the user.

6.2 Vertex finding - comparison of "geometric" scores

In order to compare performances of the various vertex reconstructors, comparison with the Monte Carlo truth is a possibility. How vertex reconstruction results are exactly compared with Monte Carlo truth, is not straightforward. In the vertex package this comparison is based on the notion of *performance estimators*. Every performance estimator is a measure for a specific aspect of the congruence between the reconstructed event and the simulated event. In vertex we find the following set of performance estimators:

- A VertexFindingEfficiencyEstimator measures the fraction of TkSimVertices which are correctly reconstructed.
- The VertexTrackAssignmentEfficiencyEstimator reports the efficiency with which tracks are correctly assigned to a RecVertex.
- The VertexTrackAssignmentPurityEstimator estimates the purity of the track to vertex assignment.
- Finally, the VertexFakeRateEstimator reports the fraction of RecVertices which can be considered fake.

These estimators form a solid base for algorithmic comparisons. In order to compare algorithms the following issues have to be addressed:

- **MC truth meddlers:** the choice of which tracks and vertices shall appear in the SimEvent is not trivial. Particles that are too short-lived to be visible should not be listed as a decay vertex. The definition of short-lived, though, may depend on the context. Development of "MC truth meddlers" has begun only recently, see Sec. B.1.2.

- **Scalar score function:** For an optimization package it is necessary to have an objective function. The objective function must be a scalar that describes the quality of the reconstruction. A score function based on a comparison with the geometric MC truth has been implemented, see Sec. B.1.1. It is a function of the performance estimators listed above. In fact it is simply a product of the performance estimators, all raised to a configurable power.

- **Tuning framework:** Given an objective function, one can tune algorithms against this function. This is done in a separate vertex package: the TuningTools – see Sec. B.1. The package is (currently) designed to optimize any single tunable parameter. For every reconstructor that is to be tuned, a separate wrapper class is written that defines how the parameter is set. The framework then allows to automatically tune the parameter and optionally compare against a baseline.

6.2.1 Results

A specific set of algorithms has been fine-tuned and compared against one another. Figs. 6.3 and 6.4 show a summary of the vertex finders in different event topologies. DetAnneal uses the default configuration of Lyon's deterministic annealing algorithm (Sec. 4.5.5). AggloIP refers to agglomerative clustering based on cluster representatives, AggloCAOH is agglomerative clustering based on the distance matrix, and AggloApex is based on apex points, see Sec. 4.4.1. KMeans denotes the k-means clusterer, Sec. 4.5.3. Because it is non-trivial to translate these results into physics performances, they shall not be discussed too deeply. It should be mentioned at this point that work has started on relating these "geometric" studies to physics performance. It can be noted that the PVR, compared with AggloIP, tends towards smaller secondary vertex finding efficiencies, but also towards lower fake rates. This seems to be a fairly general feature; a similar behavior will emerge in the more physics oriented b-tagging analysis.

6.3 Algorithmic complexity

A test was performed to evaluate the algorithmic complexity of a certain set of vertex finders. The test was run on a 2.8 GHz Intel Xeon. Fig. 6.5 shows the CPU time of the algorithms (in ms per event) as a function of the number of tracks. The number of simulated vertices in an event equals $1 + r \cdot n_{\text{tracks}}/4$, where r is a random number between 0 and 1. The HLT time budget (40 ms in 2007; $\approx$ 100 ms on a 3GHz PC) is also shown.

6.4 Vertex finding in the context of b-tagging

Two B or not two B?

William Shakespeare, "Hamlet"

One of the currently most prominent applications of secondary vertex finding is the task of b-tagging, i.e. the decision about whether or not a given jet comes from a B-meson. The b-tagging package [103] in ORCA co-evolved with the vertex package, the latter having started a little earlier. During the final stages of this thesis the b-tagging framework matured into a usable piece of code.

6.4.1 Setup

The b-tagging task can but does not have to exploit secondary vertex information. A lot of different kinds of data are usable for b-tagging. Also, a lot of methods for b-tagging have been conceived in ORCA [103]. The most general approach is one that uses a few different algorithms in a combined fashion — see the aforementioned reference for further details. The b-tagging package introduced an abstract "discriminator" value, which describes the "level of strictness" for a jet to be b-tagged. In a future release, it will be given as a probability — it will take values between zero and one. For the time being, though, it is a somewhat queer parameter that has values between 1 and infinity, and has no direct interpretation. The plots that show b-tagging efficiencies versus discriminator cuts (Fig. 6.6 and the following figures) show this discriminator parameter. The other plots in this section do not explicitly name a discriminator cut. A cut=1.0 is taken in those cases. The analyses in this section use the "BySecondaryVertex" b-tagging method. This method is known not to be optimal; the absolute b-tagging efficiencies are thus irrelevant. This method is chosen because the effects of secondary vertex reconstruction on b-tagging are more prominent.

6.4.2 Results

Fig. 6.6 together with an entire subsequent throng of figures show the results of the b-tagging performance analysis. The plots show parameter scans of the algorithms (blue lines in top row) and the default setting of the same parameter (red crosses in top row). The other six plots on the page below the top row refer to the default parameter setting (i.e. the red crosses in the top row). The efficiency versus cut and efficiency versus mass plots in the second row exploit the discriminator parameter of the b-tagging package. The discriminator is tuned against one specific algorithm! The results of the second row are thus still very preliminary, as long as optimizations are still missing. Work has begun in the b-tagging group on an automatic tuning tool. The plots in the two bottom rows, the multiplicity distribution and the mass-at-vertex distribution, show two typical variables that the b-tagging package exploits if one chooses discriminator cuts > 1.0. The more distinguished these variables are from each other in the different event topologies, the more can be gained by exploiting this information in the b-tagging algorithm. Finally let it

be mentioned that the mass and the multiplicity variables respect the track weights. This fact is particularly visible in the (fractional) multiplicity distributions of the MVF-based algorithms.

6.4.3 PVR performance in b-tagging

The PVR stands as the baseline against which all other algorithms are tested. Fig. 6.6 shows the performance profile of the method. "Default-PVR" is the DefaultPrincipalVertexReconstructor; the "PVR" series consecutively assigns values from 0.0005 to 0.50 to the parameters `VertexFitProbabilityCut`, `TrackCompatibilityCut`, and `TrackCompatibilityCutSV`. The results show a baseline that settles at $\approx$ 68% efficiency, while it maintains $\approx$ 8% and 29% fake rate in $q\bar{q}$ and $c\bar{c}$ events, respectively. The mass and multiplicity distribution plots suggest a decent separability in these two variables; this information is not taken into account in the top row.

6.4.4 The agglomerative clusterers in b-tagging

One of the main assets of the agglomerative clusterers is speed, see Fig. 6.5. Apart from CPU performance, they tend to feature both high efficiencies and fake rates, Figs. 6.8 and 6.10. An interesting combination is an agglomerative clusterer in conjunction with an MVF, Figs. 6.9 and 6.11.

6.4.5 The MultiVertexFitter in b-tagging

The MultiVertexFitter itself does not solve the pattern recognition problem; it can only work on a prior solution. Thus it makes most sense to compare solutions found by a certain vertex reconstructor with the solution found by the vertex reconstructor plus the MVF. Fig. 6.12 shows such comparisons. It shows a parameter scan of different algorithms, and plots the b-tagging efficiency versus fake rate, where fake rate is the ratio of false positives in $c\bar{c}$ or $q\bar{q}$ events. The same parameter scan is repeated with a multi vertex fit applied to the solution found by the reconstructor. The MVF is performed with the default values ($\chi^2_{\text{cut}} = 20$, $T_{\text{ini}} = 1024$, $r = 0.2$). The general trend seems to be a decrease in the fake rate that is sometimes accompanied with a small deterioration of the b-tagging efficiency.

6.4.6 Non-hierarchic clustering methods

The non-hierarchic methods are considered not mature enough to show any meaningful results. Preliminary tests so far have given results that are comparable with the hierarchic methods. Fig. 6.13 shows the results obtained so far for the super-paramagnetic clusterer (SPC), the vector quantization algorithm (VQ), and the k-means algorithm (KM).

6.4.7 The adaptive vertex reconstructor

The adaptive vertex reconstructor (AVR) has been introduced only very recently, see Sec. 4.4.2. The first impressions (Fig. 6.14) are very interesting, especially in the light of CPU performance (Fig. 6.5). A very notable feature is that the MVF does not have the "cleaning effect" that it has in combination with other algorithms. This algorithm is a good candidate for the online reconstruction task. More systematic studies are on their way.

6.4.8 A "group picture" of algorithm performances in b-tagging

Fig. 6.15 shows a "group picture", for better comparison between the different presented strategies. Note that if the best performing points from each algorithm were connected with each other, one would end up drawing a monotonously increasing line — a "maximum performance wall". Are there algorithms beyond that "performance wall"?

6.4.9 A "group picture" of the preliminary algorithms

It is also interesting to see where the immature algorithms stand, compared with the baseline. Fig. 6.16 serves this purpose.

6.4.10 Summary — vertex finders in the b-tagging context

It is clear that the task of studying the effects of vertex finders on b-tagging has only started. Before systematic studies can be performed, a few more analysis tools will have to be written. A tool that automatically tunes the discriminator ([103]) variable against variables like mass at vertex and track multiplicity still needs to be written; work on such a tool has just started. The correlations between geometric and b-tagging scores will have to be understood better. A few studies will need facilities to manipulate the Monte Carlo truth in different ways. Also, the performances of the different algorithms should not only be measured, but also qualitatively and quantitatively understood. Despite this lengthy todo list, a few features of the current implementations have already surfaced:

- The performance of the PVR baseline is quite solid. Its efficiency / fake rate ratio seems to be acceptable for many applications, only the CPU consumption of its current implementation seems to leave a large space for improvement.
- The MVF, too, seems to be a decent algorithm. Its effect tends to be a reduction in the fake rate, while often being able to maintain the b-tagging efficiency. It seems to provide a powerful "cleaning" step after having employed a vertex reconstructor for the pattern recognition task.
- The agglomerative algorithms tend to exhibit very high b-tagging efficiencies, but at the expense of high fake rates. An additional advantage is their CPU performance: they are extremely fast.

- The adaptive vertex reconstructor gives very good results at a superb CPU performance. It is an excellent candidate for online reconstruction.
- The apex point concept, prototype-based approaches, and "super-finders" could not yet be shown to increase performance. However, it is certainly premature to discard those ideas at this stage.

6.4.11 Vertex finding - correlation between geometric and b-tagging scores

In order to understand the performances of VertexReconstructors in b-tagging it is also important to establish a link between geometric scores and b-tagging performances numbers. The geometric scores are needed for in-depth understand of the algorithms, whereas the b-tagging scores connect to the physics analysis world. Fig. 6.17 shows a few plots of how these quantities correlate; it shows various kinds of vertex reconstructors run over 2500 bg and 5000 $q\bar{q}$ events, in ORCA_8_2_0. b-tagging related quantities are plotted against geometric performance estimators. Correlations between b-tagging scores and geometric scores are visible. Algorithms that assume a fixed number of reconstructed vertices fall out of the general tendency. This is documented in Fig. 6.17. It is necessary to state that the only "fixed number reconstructor" currently used in this picture is the MVF with a "BSeeder" — a very recent, experimental special purpose seeder for the MVF, that produces only one secondary vertex candidate at a certain distance along the jet axis (obtained by calorimetry information) from the primary vertex. A deeper understanding is desirable. Also the dependence on MC truth meddlers (Sec. B.1.2) is of interest. If the right MC truth meddler is implemented, and strong correlations between the "modified" geometric scores and b-tagging performance numbers are established, then the vertex finding algorithms can be analysed in great detail in a physics-oriented context. Quite some work still needs to be done to reach this level of sophistication.

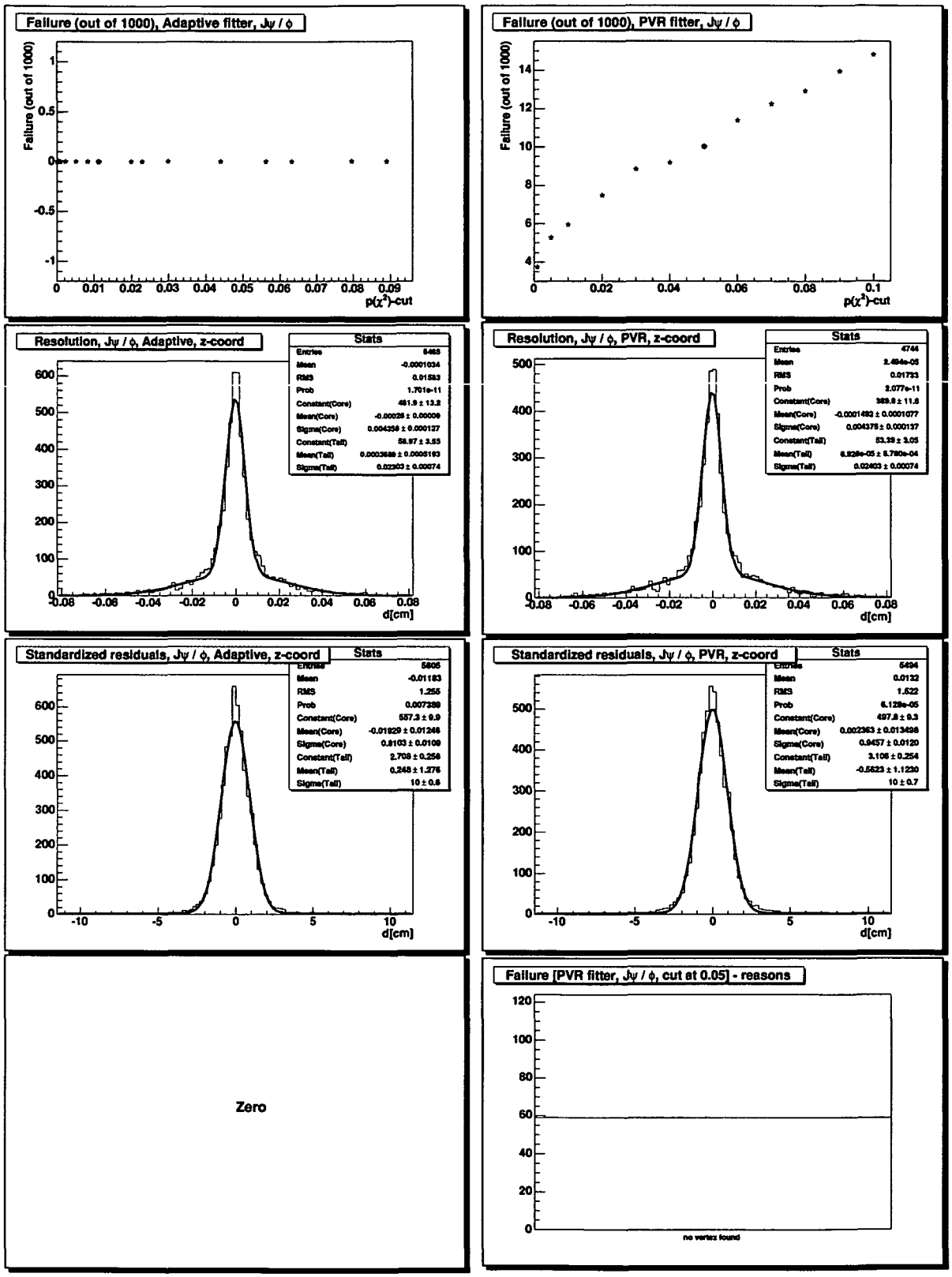


FIGURE 6.1: Comparing AVF with PVF, $J/\psi\ \phi$ channel.

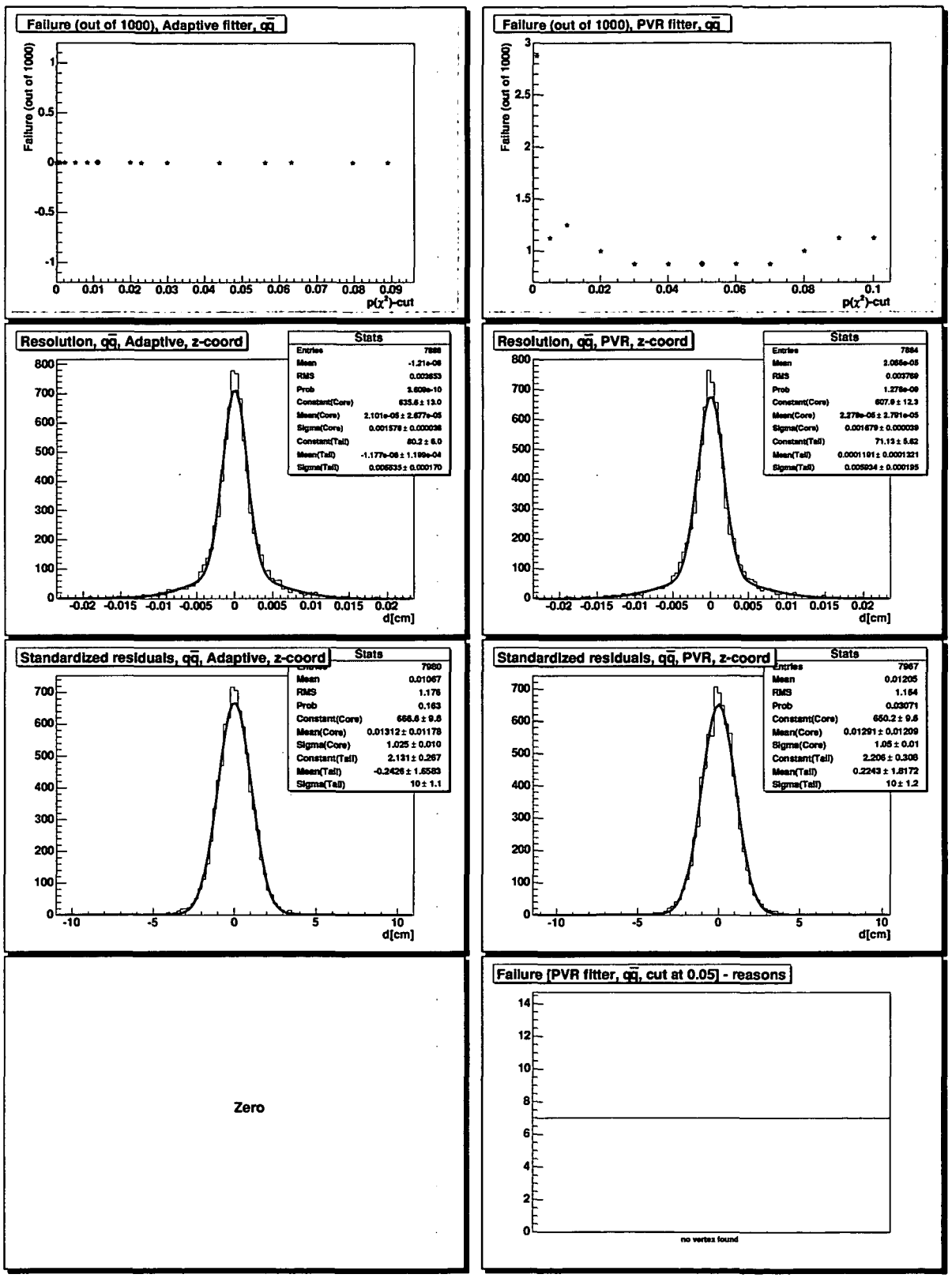


FIGURE 6.2: Comparing AVF with PVF, $q\bar{q}$ channel.

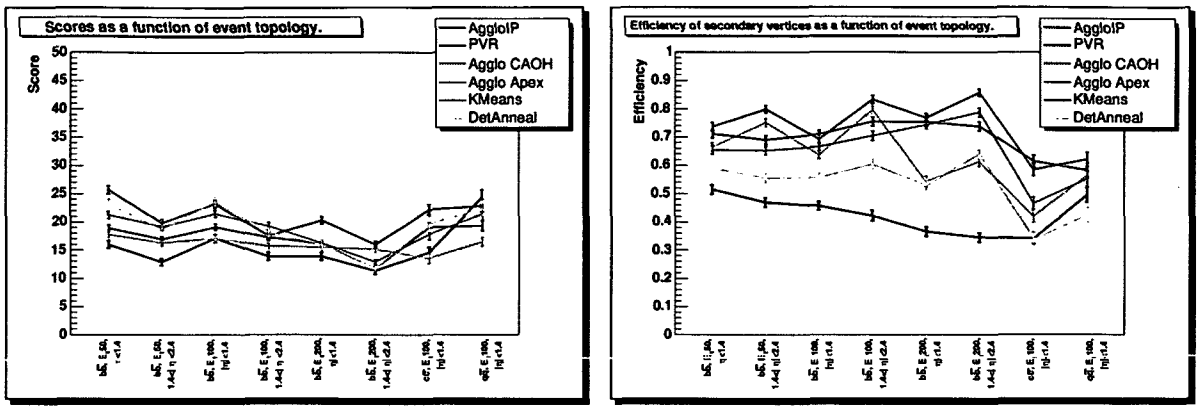


FIGURE 6.3: "Geometric" comparison of algorithms. Left: absolute scores. Right: secondary vertex finding efficiencies.

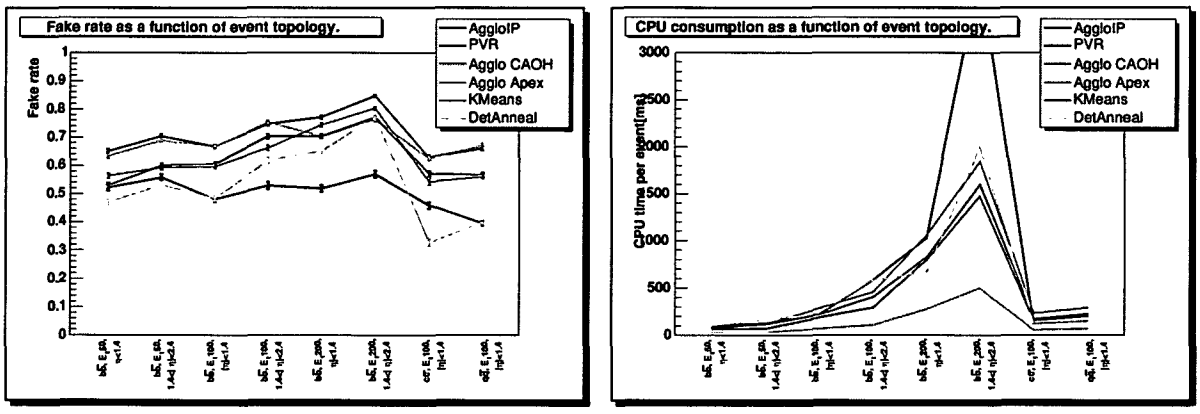


FIGURE 6.4: "Geometric" comparison of algorithms. Left: fake rates. Right: CPU consumption.

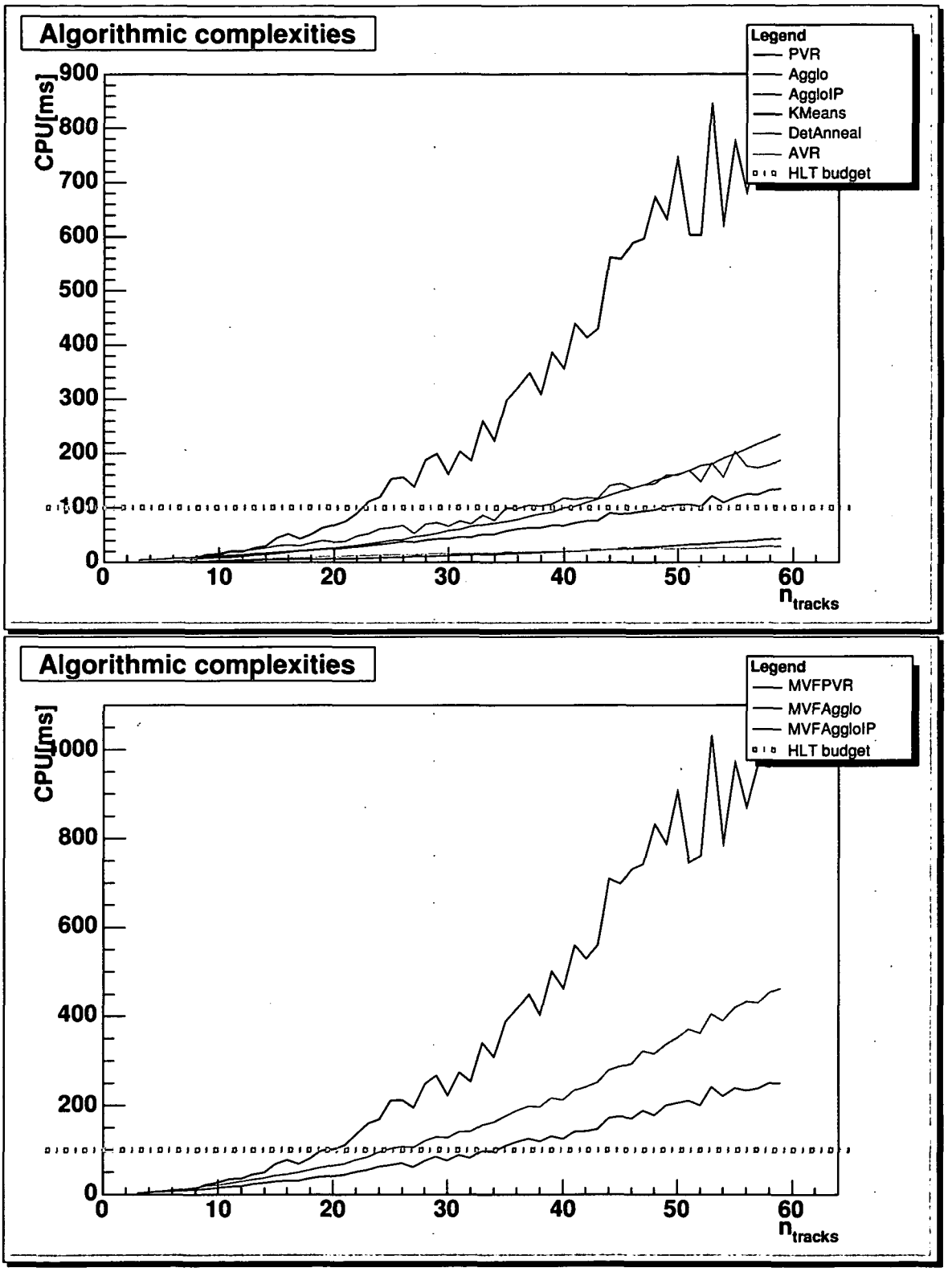


FIGURE 6.5: Algorithmic complexities, empirical tests.

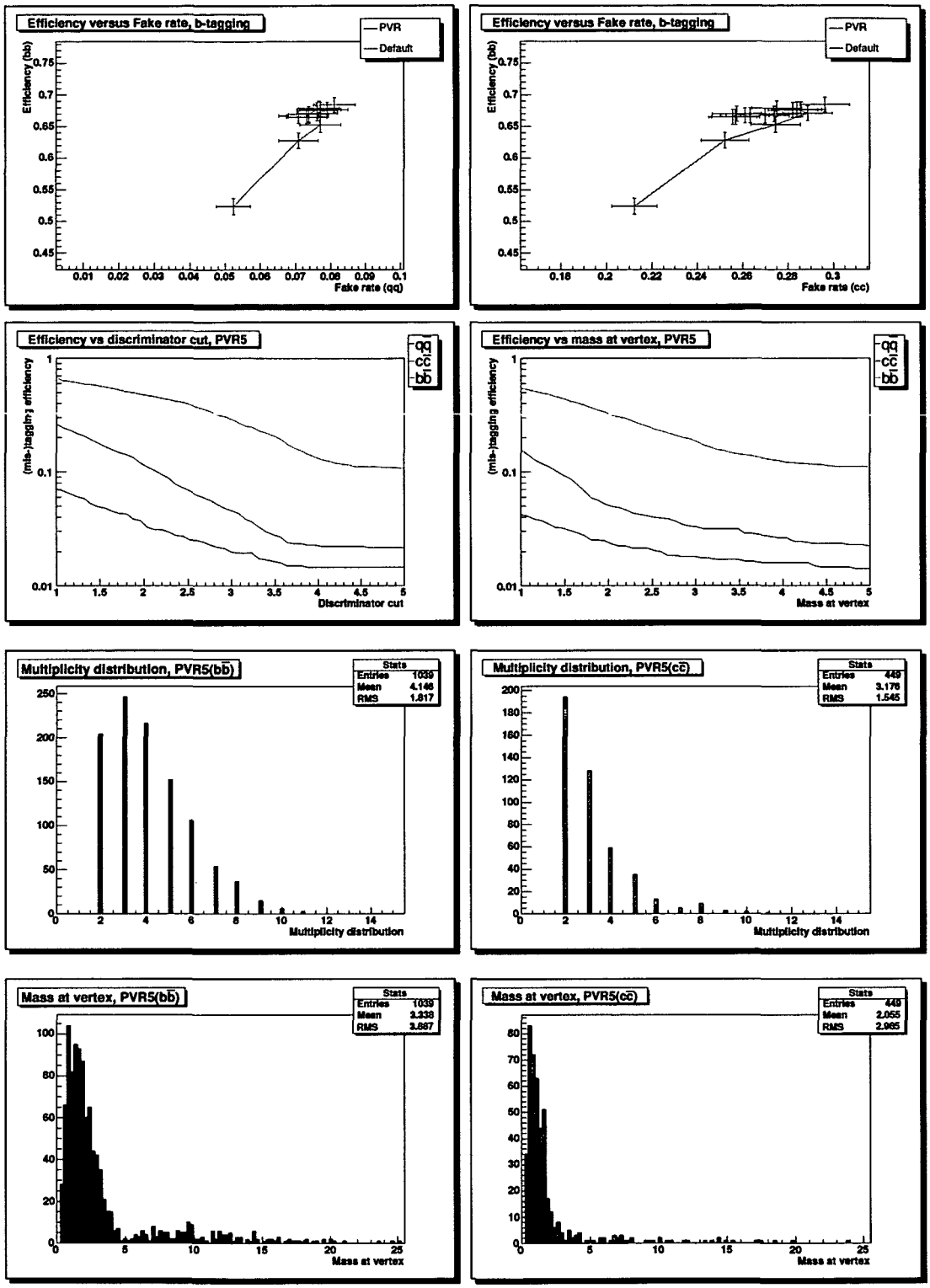


FIGURE 6.6: b-tagging efficiency, PVR. PVR parameter scan (top row), b-tagging efficiency, as a function of the discriminator cut and the vertex mass for the default PVR (second row). Differences in the track multiplicities between $b\bar{b}$ and $q\bar{q}$ (third row). Differences in the vertex masses (bottom row).

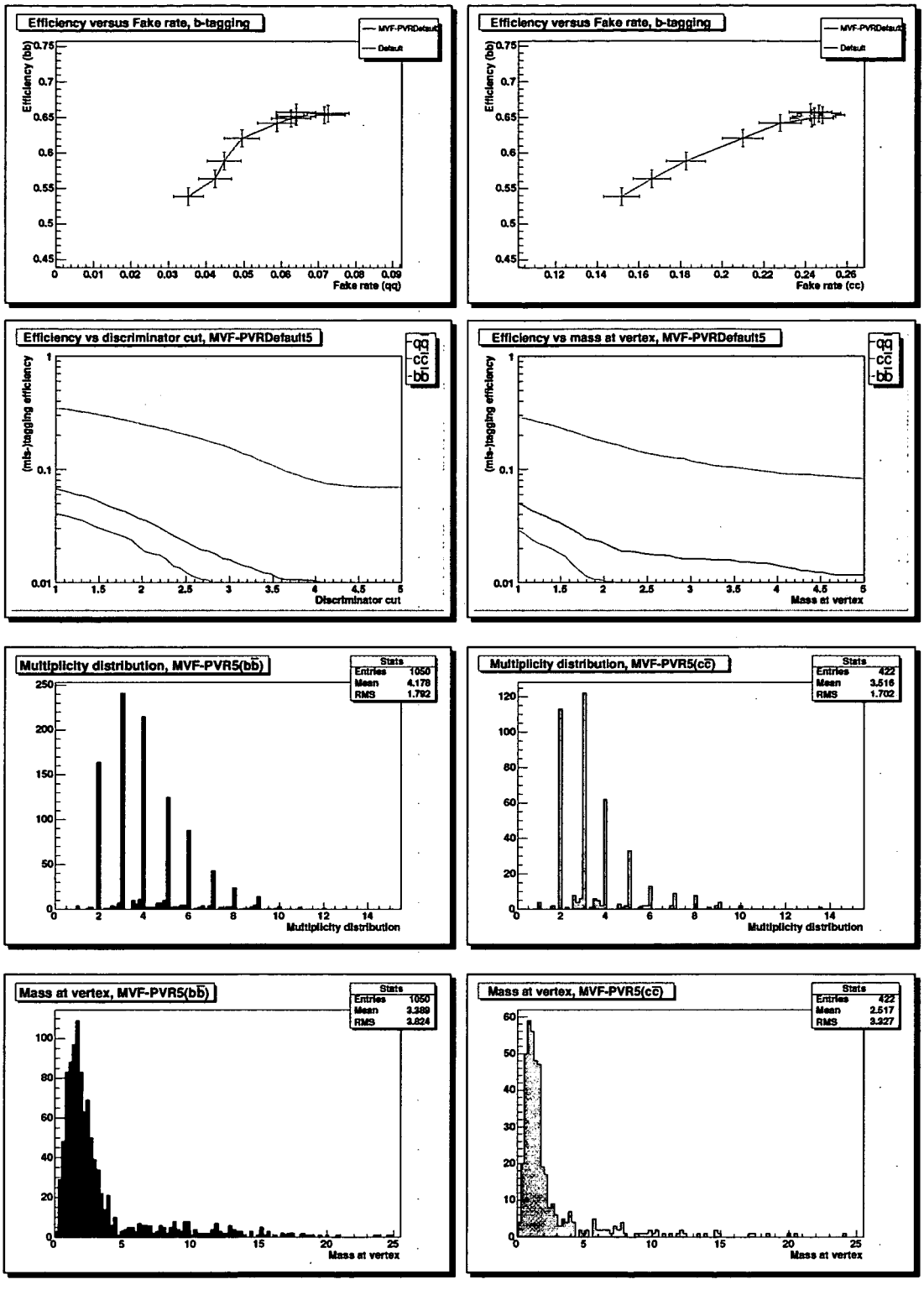


FIGURE 6.7: b-tagging efficiency, MVF-PVR. MVF-PVR parameter scan (top row), b-tagging efficiency, as a function of the discriminator cut and the vertex mass for the default MVF-PVR (second row). Differences in the track multiplicities between $b\bar{b}$ and $q\bar{q}$ (third row). Differences in the vertex masses (bottom row).

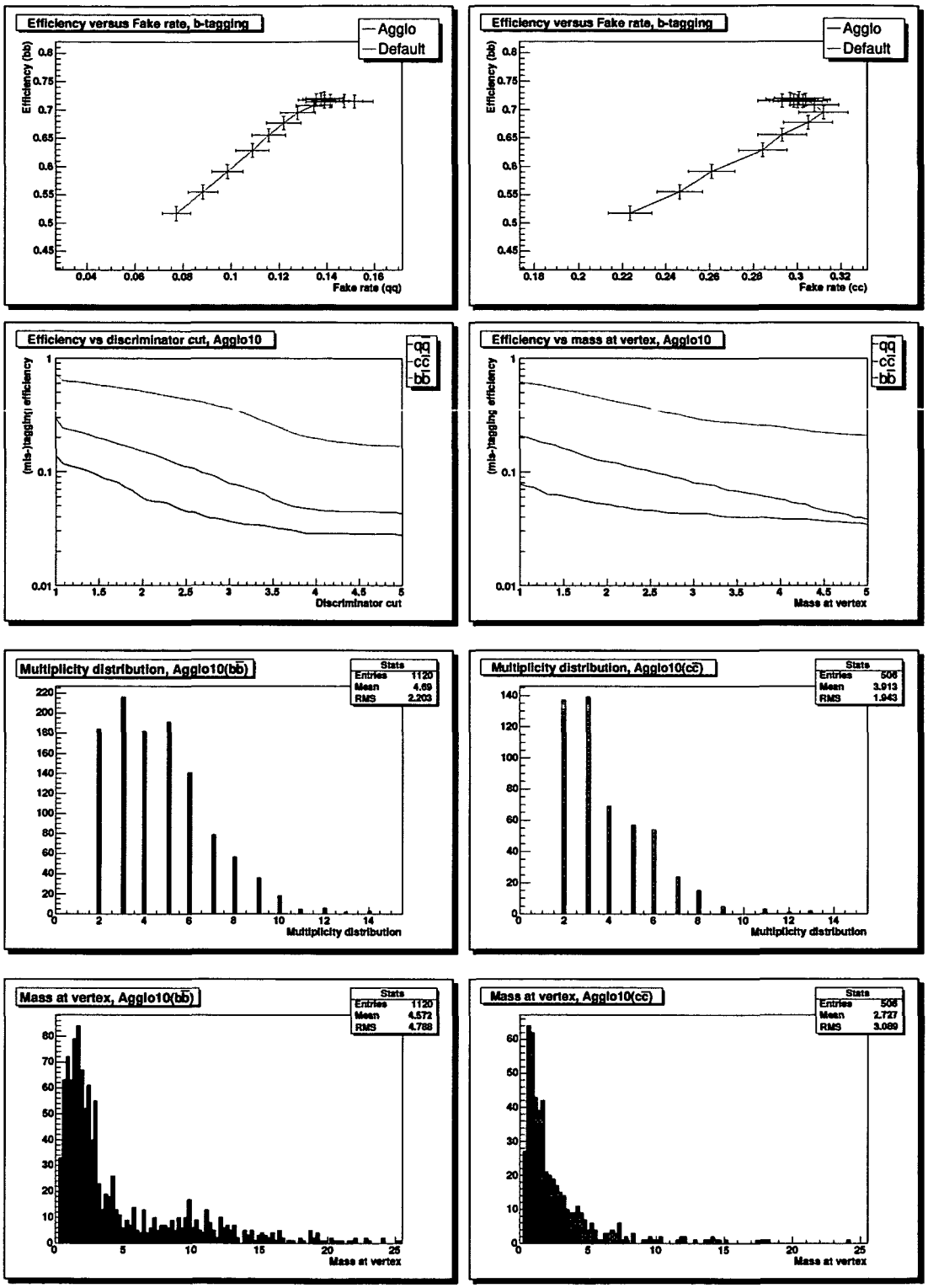


FIGURE 6.8: b-tagging efficiency, agglomerative clusterer. Parameter scan (top row), b-tagging efficiency, as a function of the discriminator cut and the vertex mass for the default Agglo (second row). Differences in the track multiplicities between $b\bar{b}$ and $q\bar{q}$ (third row). Differences in the vertex masses (bottom row).

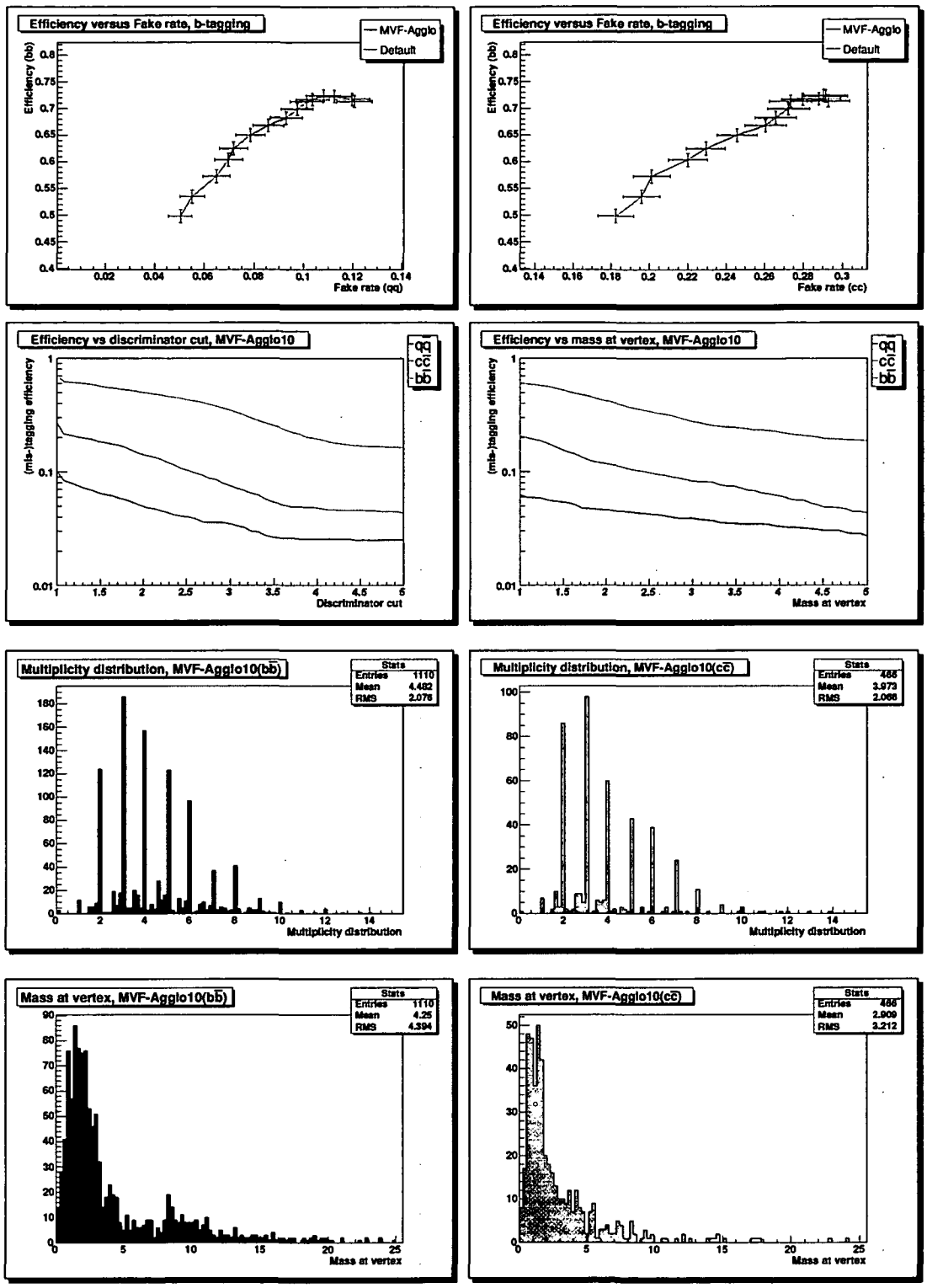


FIGURE 6.9: b-tagging efficiency, agglomerative clusterer with a subsequent MVF. Parameter scan (top row), b-tagging efficiency, as a function of the discriminator cut and the vertex mass for the default MVF-Agglo (second row). Differences in the track multiplicities between $b\bar{b}$ and $q\bar{q}$ (third row). Differences in the vertex masses (bottom row).

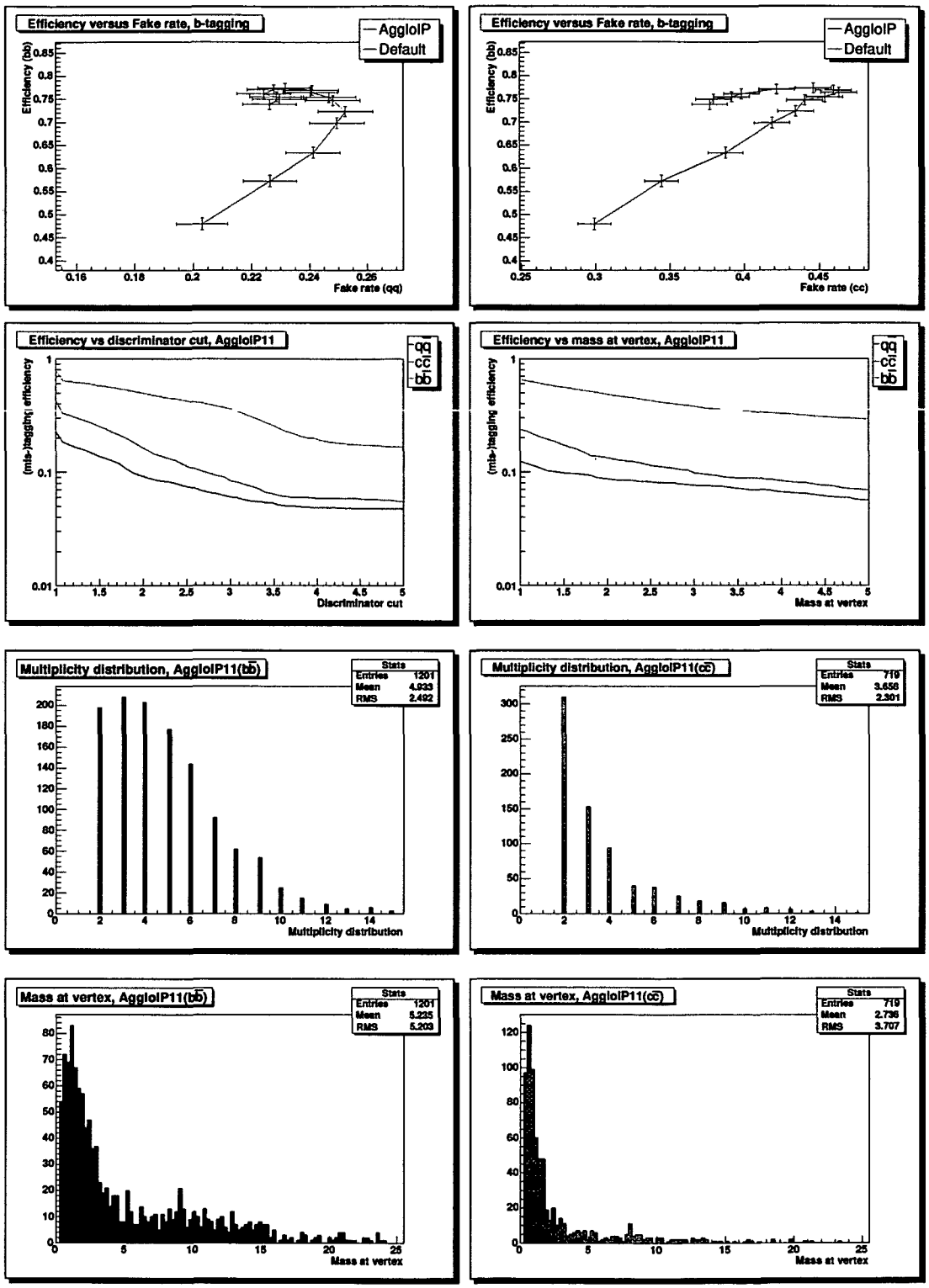


FIGURE 6.10: b-tagging efficiency, AggloIP. Parameter scan (top row), b-tagging efficiency, as a function of the discriminator cut and the vertex mass for the default AggloIP (second row). Differences in the track multiplicities between $b\bar{b}$ and $q\bar{q}$ (third row). Differences in the vertex masses (bottom row).

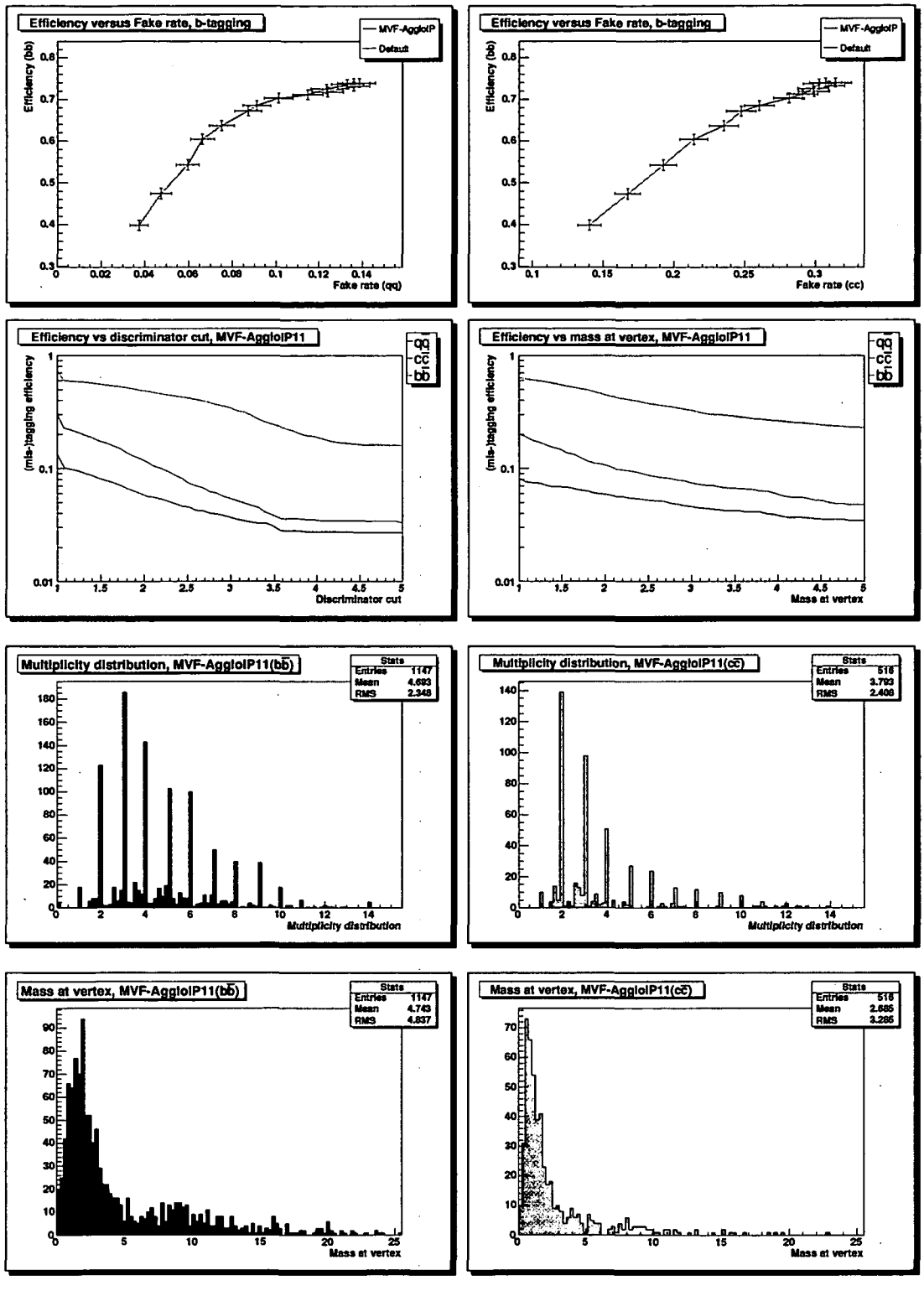


FIGURE 6.11: b-tagging efficiency, MVF-AggloIP. Parameter scan (top row), b-tagging efficiency, as a function of the discriminator cut and the vertex mass for the default MVF-AggloIP (second row). Differences in the track multiplicities between $b\bar{b}$ and $q\bar{q}$ (third row). Differences in the vertex masses (bottom row).

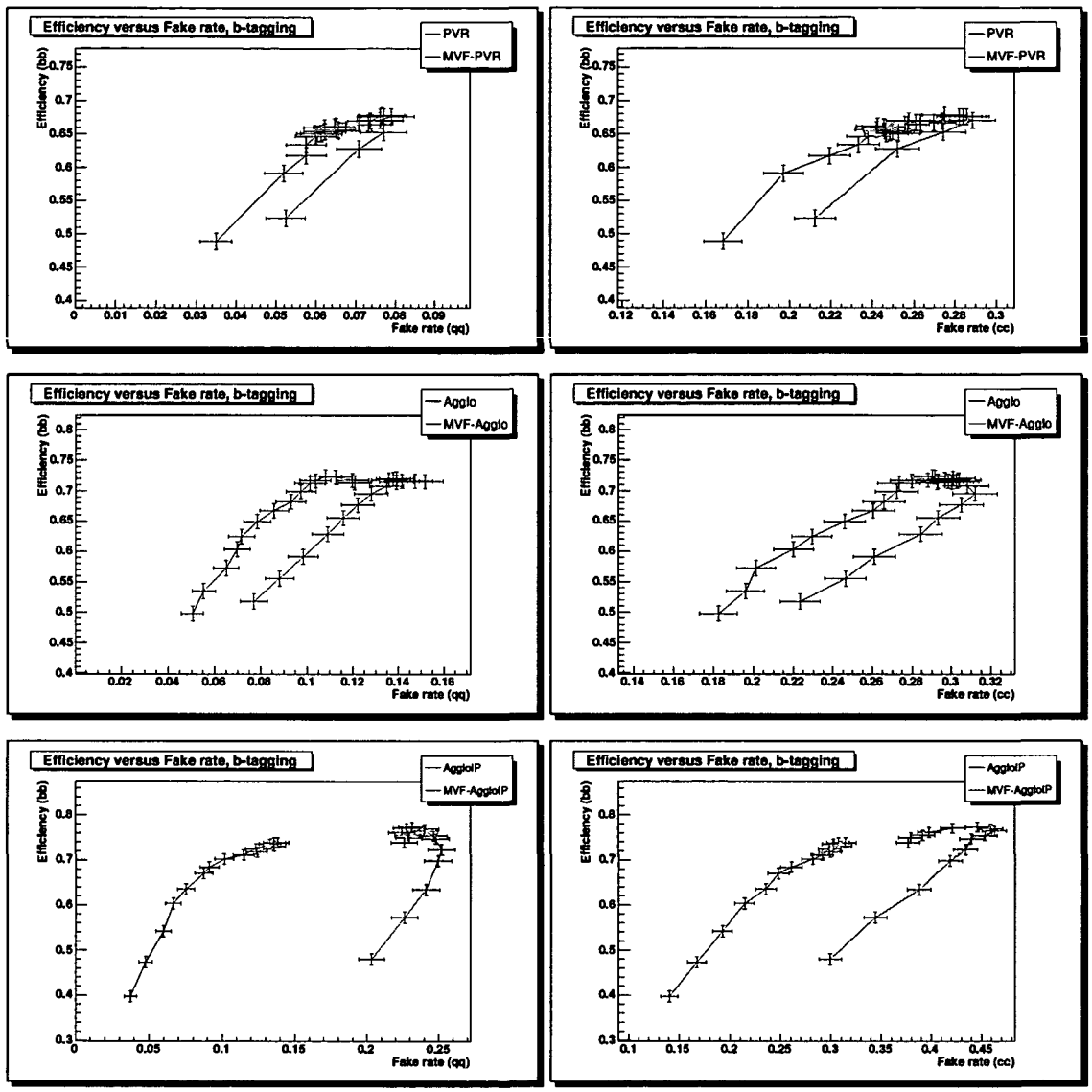


FIGURE 6.12: b-tagging efficiency, for various algorithms with and without a subsequent multi vertex fit.

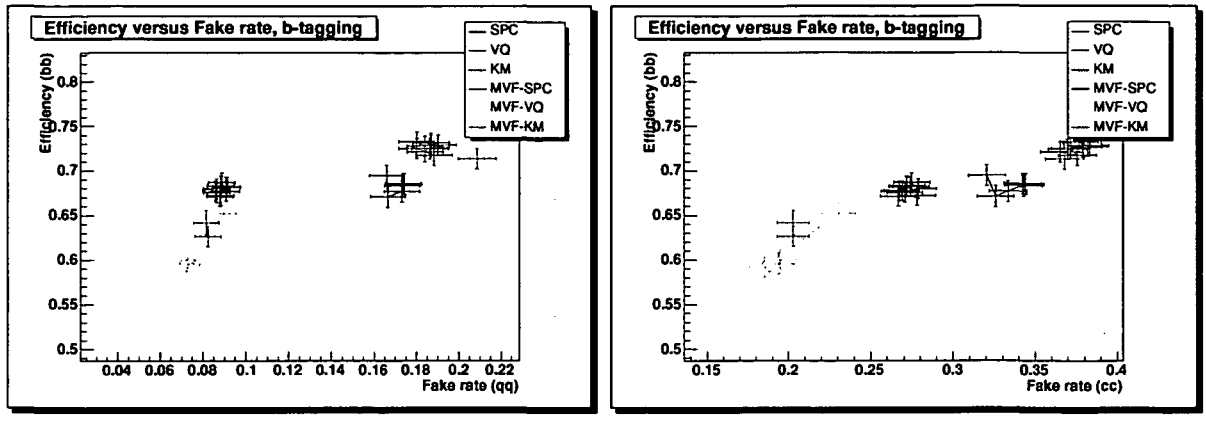


FIGURE 6.13: b-tagging efficiencies, for non-hierarchic methods — preliminary results.

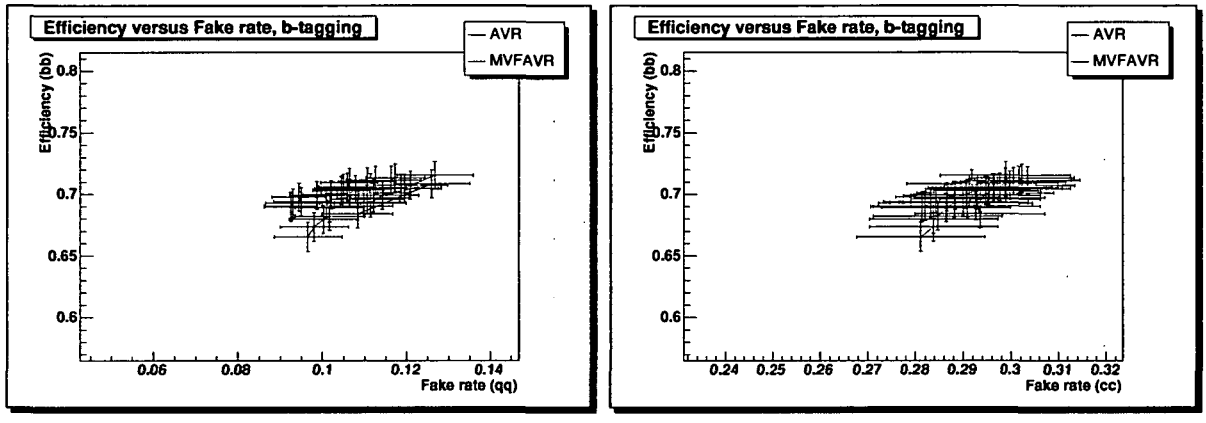


FIGURE 6.14: b-tagging efficiencies, for the AVR.

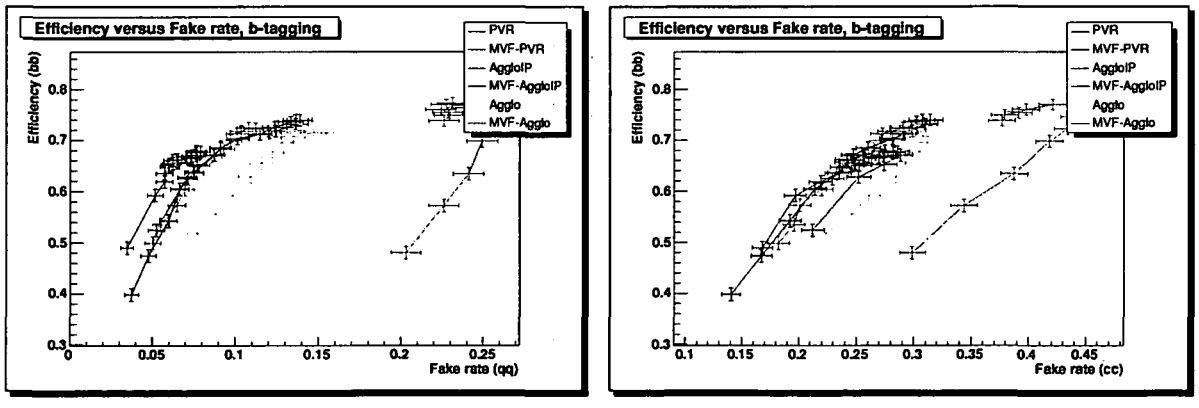


FIGURE 6.15: Group picture — all algorithms in one plot

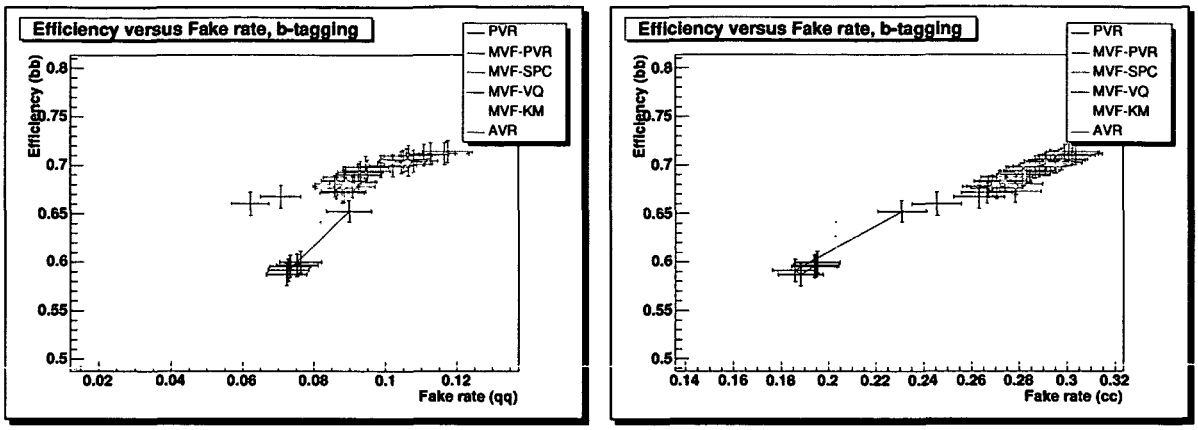


FIGURE 6.16: "Group picture" — preliminary algorithms vs baseline (PVR).

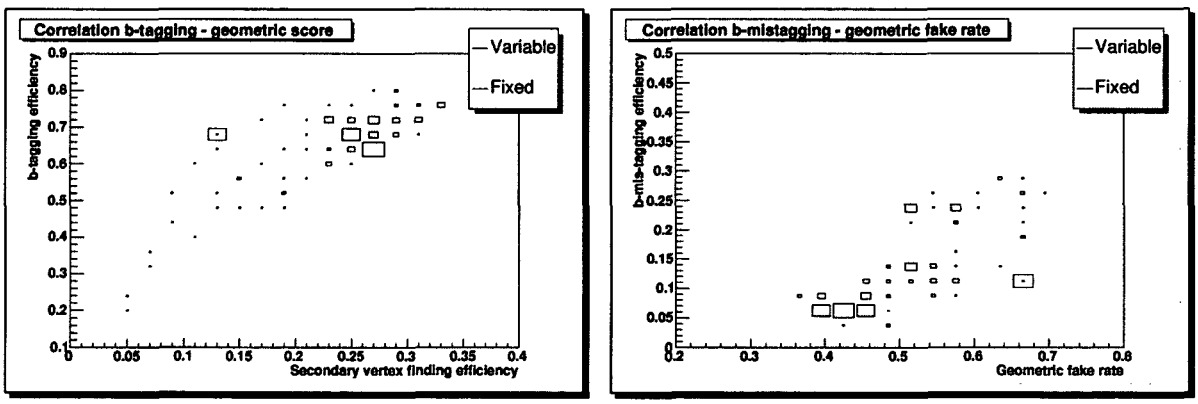


FIGURE 6.17: Correlation between geometric performance quantifiers and b-tagging scores. "Fixed" refers to algorithms that assume a fixed number of vertices (one secondary vertex and one primary vertex, in that case). "Variable" algorithms operate without such ad-hoc constraints.

CHAPTER 7

Summary and outlook

Starting from the Kalman filter, a few robustifications of least-squares vertex fitting methods have been conceived, implemented, and studied. It turned out that the **AdaptiveVertexFitter** has a wide application range; it seems a good choice for primary vertex fitting, where a lot of tracks with a lot of "contamination" point to the parton-parton collision vertex. It also seems adequate for secondary vertex fitting, where one often fits only three or four tracks into one vertex. Even in the absence of truly mis-associated tracks the adaptive method performs at least as well as more classical approaches, such as the **PrincipalVertexFitter**. In many cases the adaptive method performs significantly better. Given the current implementations, it also has the advantage of being significantly faster at higher multiplicities. It is also a more general approach; the user is given an additional piece of information, the track weights. How much the user can benefit from this extra information, is difficult to predict and shall not be attempted here.

All vertex fitters, especially the non-linear ones, require an initialization point. Many *LinearizationPointFinders* have been designed, implemented, and studied. A very good general purpose method that covers a wide range of physics applications has been identified. This problem, it seems, can be seen as being solved.

In the field of vertex finding an entire algorithm zoo has been designed and implemented. The performance tests in this field have not yet progressed as far as in the case of vertex fitting; still it has already matured enough to merit a few comments. The agglomerative cluster finders work satisfactorily. One very recent addition, the **AdaptiveVertexReconstructor** could also be shown to exhibit good efficiency versus fake rate ratios, with an excellent CPU performance. It will be suggested as the default online HLT algorithm. The **MultiVertexFitter** has been implemented and tested. It shares its theoretical background with the **AdaptiveVertexFitter** and seems to be an adequate algorithm for a "clean-up" step: in combination with other vertex reconstructors it exhibits smaller fake rates. It is somewhat unfortunate that the "meta-algorithms", such as the **BestSolution-**

Finder and the VotingFinder do not show significant improvement over the baseline. Also, the non-hierarchic methods have not yet had any successful impact. It is, on the other hand, still too early to discard these ideas entirely; deeper insight is required. A comprehensive understanding of why each algorithm performs how well under what circumstances is still missing.

The task of computing the smallest distance between two helices has seen a major improvement. In the context of this thesis it was possible to develop a simple algorithm that improves results dramatically.

A few of the tools developed in the context of this thesis seem to thrive and take on lives on their own. The algorithmic tuning tool still lacks a generalization of the score function. Besides that it seems to turn into a basic standard tool of the vertexing package. Still more successful (in terms of usage) is the VertexFastSim package. By now, practically all of the vertex package's code verification makes use of the package. Even a few official studies make use of the fine-grained control over the event data that this package allows for. The data harvester is a more recent development. It, too, seems to develop into a direction that has not been anticipated when the harvester has first been conceived. It is now a highly usable non-intrusive debugging tool that shields various persistency technologies from the user. Still, at this point, it remains a special purpose debugging tool, although the future plans are ambitious. Finally a small and simple visualization tool has been written for this thesis. It is a special purpose tool and does not even pretend to compete with big, general-purpose artefacts like IGUANA. Especially the last two tools have the potential of being used outside of CMS as well. In both cases special interest by other groups has already been declared.

Another idea that has been born in the process of this thesis is the creation of a detector independent vertex reconstruction toolkit [76, 77]. The big advantage of the task of vertex reconstruction is its high level of abstraction; it is fairly detector independent. The only detector-dependent issues can easily be hidden in Propagator and MagneticField objects. The number of *connector* classes that need to be interfaced to the environment is thus fairly small. A part from the aforementioned Propagators and MagneticField, one also needs to define RecTracks and RecVertices. These should already be sufficient, apart from "framework infrastructure", such as configuration and logging utilities. A similar project is "RecPack" [98]; it has the much more ambitious goal of providing a detector independent track reconstruction toolkit.

CHAPTER A

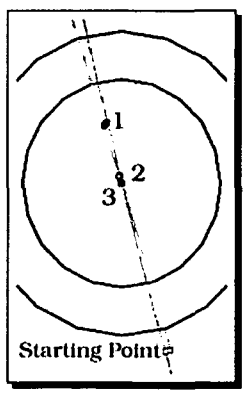
Various algorithms

This chapter is dedicated to small "math snippets" that have not found an appropriate place in one of the main chapters.

A.1 Minimal distance between two helices

A very fast method of finding the minimal distance between two helices is essential for many different clustering algorithms that work on the distance matrix as well as for many linearization point finders, including the default method.

Finding the local minima



For the problem of finding a local minimum a Newtonian procedure that finds the zeros of the distance function's gradient is used. A general parametrized helix can be given by [52]

$$\mathbf{H}(\phi_H;\lambda_H,h,{\phi^0}_H,\mathbf{H_0}) \equiv \mathbf{H_0}+\mathbf{h} = \mathbf{H_0}+h\begin{pmatrix}\sin\phi_H-\sin\phi_H^0\\ -\cos\phi_H+\cos\phi_H^0\\ \tan\lambda_H(\phi_H-\phi_H^0)\end{pmatrix} \tag{A.1}$$

where we have used the signed projected radius of the helix $h=-\frac{|\mathbf{p}_H|\cos\lambda}{\kappa qB}$; $\mathbf{H_0}$ contains the coordinate values at $\phi_H=\phi_H^0$, and $\lambda_H=\frac{\pi}{2}-\vartheta$ is the slope angle. Our helix is defined by a point in space, a momentum vector, and the "charge" (that is ±1). The point in space is represented by $\mathbf{H}_0$, the momentum vector goes into (A.1) the following way: Solving the Lorentz

force equation $f = \frac{d\mathbf{p}}{dt} = c^2\kappa q\mathbf{v}(t) \times \mathbf{B}(\mathbf{x}(t))$ results in:

$$\phi_H = \phi_H^0 - \frac{c^2 q_H \kappa B}{m\gamma} t \tag{A.2}$$

The initial momenta $\mathbf{p}_H$ can be calculated easily:

$$\mathbf{p}_H = m\gamma \left.\frac{\mathrm{d}\mathbf{H}(\phi_H(t))}{\mathrm{d}t}\right|_{\phi_H=\phi_H^0} = -q_H B c^2 \kappa h \begin{pmatrix} \cos\phi_H^0 \\ \sin\phi_H^0 \\ \tan\lambda_H \end{pmatrix} \tag{A.3}$$

Solving for $\sin\phi_H^0$ et al:

$$\cos\phi_H^0 = -\frac{p_{x,H}}{q_H B c^2 \kappa h},\ \sin\phi_H^0 = -\frac{p_{y,H}}{q_H B c^2 \kappa h},\ \tan\lambda_H = -\frac{p_{z,H}}{q_H B c^2 \kappa h} \tag{A.4}$$

h is not determined yet. Solving for h, and plugging in yields:

$$\begin{aligned} \lambda_H &= \arcsin\left(\frac{p_{z,H}}{|\mathbf{p}_H| c^2}\right) \\ h &= \frac{|\mathbf{p}_H|}{\kappa q B}\sqrt{1 - \left(\frac{p_{z,H}}{|\mathbf{p}_H| c^2}\right)^2} \end{aligned} \tag{A.5}$$

A.1.1 Newton-Kantorowitsch method

We denote a second helix with **G**:

$$\mathbf{G}(\phi_G; \lambda_G, g, \phi^0{}_G, \mathbf{G_0}) \equiv \mathbf{G_0} + \mathbf{g} = \mathbf{G_0} + g \begin{pmatrix} \sin\phi_G - \sin\phi_G^0 \\ -\cos\phi_G + \cos\phi_G^0 \\ \tan\lambda_G(\phi_G - \phi_G^0) \end{pmatrix} \tag{A.6}$$

The squared distance between those two is then:

$$\mathbf{d}^2(\mathbf{H}(\phi_H), \mathbf{G}(\phi_G)) = (\mathbf{H_0} + \mathbf{h} - \mathbf{G_0} - \mathbf{g})^2 \tag{A.7}$$

A local minimum certainly has to fulfill:

$$0 = \frac{\partial \mathbf{d}^2(\mathbf{H}(\phi_H), \mathbf{G}(\phi_G))}{\partial \phi_H} = 2\mathbf{d}^{\mathbf{T}} \cdot \frac{\mathrm{d}\mathbf{h}}{\mathrm{d}\phi_H} \tag{A.8}$$

$$0 = \frac{\partial \mathbf{d}^2(\mathbf{H}(\phi_H), \mathbf{G}(\phi_G))}{\partial \phi_G} = -2\mathbf{d}^{\mathbf{T}} \cdot \frac{\mathrm{d}\mathbf{g}}{\mathrm{d}\phi_G} \tag{A.9}$$

This translates into the following two constraints:

$$\begin{aligned} &\cos\phi_H \left[H^{0x} - G^{0x} + h(\sin\phi_H - \sin\phi_H^0) - g(\sin\phi_G - \sin\phi_G^0)\right] + \\ &\sin\phi_H \left[H^{0y} - G^{0y} + h(-\cos\phi_H + \cos\phi_H^0) - g(-\cos\phi_G + \cos\phi_G^0)\right] + \\ &\tan\lambda_H \left[H^{0z} - G^{0z} + h\tan\lambda_H(\phi_H - \phi_H^0) - g\tan\lambda_G(\phi_G - \phi_G^0)\right] = 0 \end{aligned} \tag{A.10}$$

$$\begin{aligned}
&\cos\phi_G\left[H^{0x}-G^{0x}+h(\sin\phi_H-\sin\phi_H^0)-g(\sin\phi_G-\sin\phi_G^0)\right]+\\
&\sin\phi_G\left[H^{0y}-G^{0y}+h(-\cos\phi_H+\cos\phi_H^0)-g(-\cos\phi_G+\cos\phi_G^0)\right]+\\
&\tan\lambda_G\left[H^{0z}-G^{0z}+h\tan\lambda_H(\phi_H-\phi_H^0)-g\tan\lambda_G(\phi_G-\phi_G^0)\right]=0
\end{aligned} \tag{A.11}$$

These are transcendent equations of the following type:

$$\begin{aligned}
z_1 \equiv\quad & \cos\phi_H(a-g\sin\phi_G)+\sin\phi_H(b+g\cos\phi_G)+\\
& c_1\phi_H+d_1\phi_G+e_1=0\\
z_2 \equiv\quad & \cos\phi_G(a+h\sin\phi_H)+\sin\phi_G(b-h\cos\phi_H)+\\
& c_2\phi_G+d_2\phi_H+e_2=0
\end{aligned} \tag{A.12}$$

The coefficients correspond to the parameters:

$$\begin{aligned}
a &= H^{0x}-G^{0x}+g\sin\phi_G^0-h\sin\phi_H^0\\
b &= H^{0y}-G^{0y}-g\cos\phi_G^0+h\cos\phi_H^0\\
c_1 &= h(\tan\lambda_H)^2,\ c_2=-g(\tan\lambda_G)^2\\
d_1 &= -g\tan\lambda_G\tan\lambda_H,\ d_2=h\tan\lambda_G\tan\lambda_H\\
e_1 &= \tan\lambda_H\left(H^{0z}-G^{0z}-h\phi_H^0\tan\lambda_H+g\phi_G^0\tan\lambda_G\right)\\
e_2 &= \tan\lambda_G\left(H^{0z}-G^{0z}-h\phi_H^0\tan\lambda_H+g\phi_G^0\tan\lambda_G\right)
\end{aligned} \tag{A.13}$$

We shall solve the equations using a *Newton-Kantorowitsch* approximation. The following equation has to be solved iteratively:

$$0=\mathbf{z}\begin{pmatrix}\phi_{H,i-1}\\ \phi_{G,i-1}\end{pmatrix}+\mathbf{A}(\phi_{H,i-1},\phi_{G,i-1})\left(\begin{pmatrix}\phi_{H,i}\\ \phi_{G,i}\end{pmatrix}-\begin{pmatrix}\phi_{H,i-1}\\ \phi_{G,i-1}\end{pmatrix}\right) \tag{A.14}$$

where $\mathbf{A}$ is the Jacobian matrix to $\mathbf{z}$, its elements being:

$$\begin{aligned}
A_{11} &= \frac{\partial z_1}{\partial\phi_H}=-\sin\phi_H(a-g\sin\phi_G)+\cos\phi_H(b+g\cos\phi_G)+c_1\\
A_{22} &= \frac{\partial z_2}{\partial\phi_G}=-\sin\phi_G(a+h\sin\phi_H)+\cos\phi_G(b-h\cos\phi_H)+c_2\\
A_{12} &= \frac{\partial z_1}{\partial\phi_G}=-g\cos\phi_H\cos\phi_G-g\sin\phi_H\sin\phi_G+d_1\\
A_{21} &= \frac{\partial z_2}{\partial\phi_H}=h\cos\phi_H\cos\phi_G+h\sin\phi_H\sin\phi_G+d_2
\end{aligned} \tag{A.15}$$

z_1 and z_2 denote the components of $\mathbf{z}=\begin{pmatrix}z_1\\ z_2\end{pmatrix}$. The iteration can be explicitly solved for our new $\phi_{H,i},\phi_{G,i}$:

$$\begin{aligned}
\phi_{H,i} &= \phi_{H,i-1}+\Delta\phi_H=\phi_{H,i-1}-\frac{z_1A_{22}-z_2A_{12}}{\det A}\\
\phi_{G,i} &= \phi_{G,i-1}+\Delta\phi_G=\phi_{G,i-1}-\frac{z_2A_{11}-z_1A_{21}}{\det A}
\end{aligned} \tag{A.16}$$

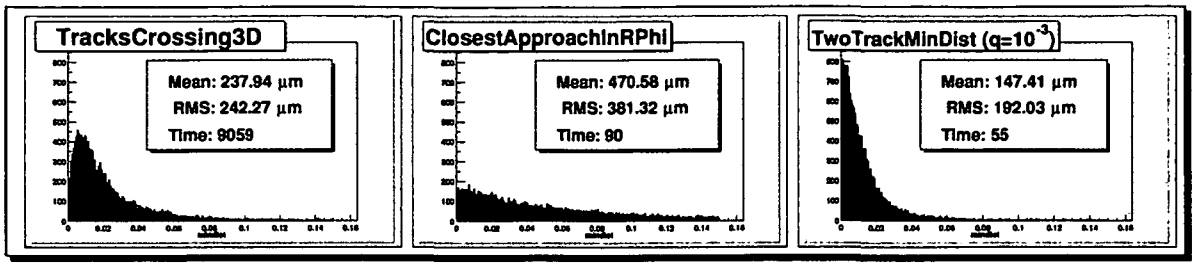


FIGURE A.1: Different implementations of ClosestApproachOfHelices in comparison.

A.1.2 The final recipe

... is very simple. The iteration Eq. (A.14) has to be solved, its solution being Eq. (A.16). So we start with some initial values, for instance $\phi_{H,0} = \phi_H^0$, $\phi_{G,0} = \phi_G^0$. The coefficients Eq. (A.4), Eq. (A.13) are calculated. Then, in an iteration, one can compute $\mathbf{A}$ (Eq. (A.15)) and $\mathbf{z}$ (Eq. (A.12)), new $\phi_{H,G}$ are obtained, that can again be used for a new $\mathbf{A}$, $\mathbf{z}$, ...

A.1.3 Comparison with other algorithms

This algorithm compares favorably with the other algorithms, see Fig. A.1. The figure shows a comparison with tracks from VertexGun events. TwoTrackMinimumDistance (the method presented in this section) is both faster and more precise than the previously used algorithms. Note that in order to enhance the algorithm's numerical stability, the following strategy has been implemented:

- Try to compute the minimum distance with TwoTrackMinimumDistance, using the innermostStates() as the initial points.
- If this fails (i.e. if the algorithm does not converge), we use the result of ClosestApproachInRPhi as the initialization for another run of TwoTrackMinimumDistance.

This strategy benefits from both the robustness of ClosestApproachInRPhi and the precision and speed of TwoTrackMinimumDistance. It has been used extensively since a long time; so far we are not aware of any problems with this approach.

A.2 Rotation matrix → rotation axis

For visualization of the 3x3 covariance matrices of points SoEllipsoid of the hepvis classes [6] were used. These classes do not receive covariance matrices as their input. Rather they expect the eigenvalues of the matrix in its diagonal form, together with a rotation axis and a rotation angle, as is indeed the standard in Coin3D [4], and, in fact, also in VRML. As simple as the transformation is, there are a few caveats that accompany the computation. Thus, we shall briefly describe the algorithm that is employed in the visualization tool.

We start with an arbitrary covariance matrix $\mathbf{V}$, and assume that the diagonalization of $\mathbf{V}$ is already accounted for within a general-purpose linear algebra framework;

$$\mathbf{V} = \mathbf{U}\mathbf{D}\mathbf{U}^{\mathbf{T}}. \tag{A.17}$$

The first task, finding the eigenvalues, is already accomplished.

$$\text{eigenvalues} = \mathbf{D}_{1,1}, \mathbf{D}_{2,2}, \mathbf{D}_{3,3}$$

We demand that the determinant of the rotation matrix is $+1$. If it is -1 any two rows of $\mathbf{U}$ (and, likewise, of $\mathbf{D}$) can be swapped in order to change the sign of the determinant. The rotation matrix $\mathbf{U}$ (with $\det(\mathbf{U}) = +1$) has to be described as a rotation axis $\vec{h}$ and angle α. $\vec{h}$ is – in spaces with an odd number of dimensions – the eigenvector to the eigenvalue of 1. $\vec{h}$ is thus determined by:

$$(\mathbf{U} - \mathbf{I})\,\vec{h} = 0 \tag{A.18}$$

The matrix $\mathbf{U}-\mathbf{I}$ is of rank 2. This is equivalent to the fact that the length of $\vec{h}$ is arbitrary. As one can see from A.18, $\vec{h}$ has to be orthogonal to any row of the matrix $\mathbf{U} - \mathbf{I}$. Hence we can take any one non-zero row

$$\vec{b}_i = (U_{i,1} - \delta_{i,1}, U_{i,2} - \delta_{i,2}, U_{i,3} - \delta_{i,3}) \tag{A.19}$$

and rotate it with the rotation matrix $\mathbf{U}$:

$$\vec{b}'_i = \mathbf{U}\vec{b}_i. \tag{A.20}$$

h is now simply

$$\vec{h} = \frac{\vec{b}_i \times \vec{b}'_i}{|\vec{b}_i \times \vec{b}'_i|}. \tag{A.21}$$

Its associated angle can be determined by computing "how far" $\vec{b}'_i$ has been rotated with respect to $\vec{b}_i$:

$$\alpha = \operatorname{acos}\left(\frac{\vec{b}_i \cdot \vec{b}'_i}{|\vec{b}_i||\vec{b}'_i|}\right) \tag{A.22}$$

Note that the procedure fails (only) in the case that $\mathbf{U}$ is the identity matrix.

A.3 Track error transformation

Per'i'gee n. (Astron.) That point, in the orbit of the moon or other body orbiting the earth, which is nearest to the earth; — opposed to apogee. It is sometimes, but rarely, used of the nearest points of bodies not orbiting the earth, such as of a comet, a planet, etc. Called also epigee, epigeum.

Webster's Online Dictionary

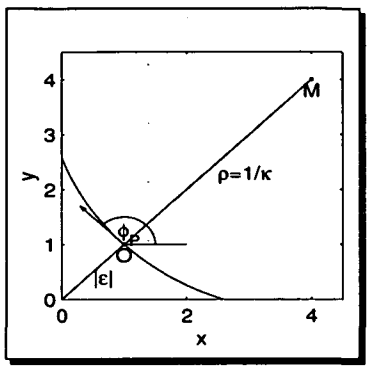


FIGURE A.2: Perigee parameters, $R-\phi$ projection only

Usually the perigee parametrization of the track has computational advantages over the global Euclidean parameters. Its definition point P is usually defined as the point of closest approach to the beamline, which, in our case, is the z axis. With this pivot point a track is parametrized [27] as $t_{\text{perigee}} = (\epsilon, z_P, \theta, \phi_P, \kappa)^T$. If O denotes the point of closest approach to the track at the z axis, then P is defined by the angle ϕ_P and the norm of ϵ, where ϵ is the signed distance $\overline{OP}$. $\text{sgn}(\epsilon)$ is defined to be $+1$ if the angle between $\overline{OP}$ and the track direction is $+\pi/2$. z_P is the z offset of P, θ is the polar angle of the trajectory with respect to the z-axis, and κ is the signed curvature in the $x-y$ projection, $\kappa = \frac{Bkq}{p_t}$ The sign of κ is positive if the trajectory is anticlockwise. Let further L denote the projected path length. With these parameters a track is described as

$$\vec{x}(\lambda) = \begin{pmatrix} \epsilon \sin\phi_P + \lambda\cos\phi_P - \frac{L^2\kappa}{2}\sin\phi_P \\ -\epsilon\cos\phi_P + \lambda\sin\phi_P + \frac{L^2\kappa}{2}\cos\phi_P \\ z_P + \lambda\cot\theta \end{pmatrix} \tag{A.23}$$

For visualization, these parameters had to be expressed in terms of the global Euclidean parameters (x, y, z, p_x, p_y, p_z). Thus, with the auxiliary variables

$$Q \equiv x\frac{p_x}{p_t} + y\frac{p_y}{p_t} \tag{A.24}$$

$$R \equiv y\frac{p_x}{p_t} - x\frac{p_y}{p_t} \tag{A.25}$$

the perigee parameters can be expressed as (see [27]):

$$\epsilon = -R - Q^2\kappa/2 \tag{A.26}$$

$$z_P = z - Q(1 - R\kappa)\frac{p_z}{p_t} \tag{A.27}$$

$$\phi_P = \phi - Q(1 - R\kappa)\kappa \tag{A.28}$$

If one also wants to visualize the transversal track errors $\sigma(\epsilon_P)$ and $\sigma(z_P)$, then one needs:

$$\mathrm{var}(p_t) = \left(\frac{px}{pt}\right)^2 \mathrm{var}(p_x) + \left(\frac{py}{pt}\right)^2 \mathrm{var}(p_y) \tag{A.29}$$

$$\mathrm{var}(\kappa) = \left(\frac{B\kappa q}{pt^2}\right)^2 \mathrm{var}(p_t) \tag{A.30}$$

$$\begin{aligned}\mathrm{var}(Q) = {} & \left(\frac{p_x}{p_t}\right)^2 \mathrm{var}(x) + \left(\frac{x}{p_t}\right)^2 \mathrm{var}(p_x) + \left(\frac{xp_x}{p_t}\right)^2 \mathrm{var}(p_t) + \\ & + \left(\frac{p_y}{p_t}\right)^2 \mathrm{var}(y) + \left(\frac{y}{p_t}\right)^2 \mathrm{var}(p_y) + \left(\frac{yp_y}{p_t}\right)^2 \mathrm{var}(p_t)\end{aligned} \tag{A.31}$$

$$\begin{aligned}\mathrm{var}(R) = {} & \left(\frac{p_x}{p_t}\right)^2 \mathrm{var}(y) + \left(\frac{y}{p_t}\right)^2 \mathrm{var}(p_x) + \left(\frac{yp_x}{p_t}\right)^2 \mathrm{var}(p_t) + \\ & + \left(\frac{p_y}{p_t}\right)^2 \mathrm{var}(x) + \left(\frac{x}{p_t}\right)^2 \mathrm{var}(p_y) + \left(\frac{xp_y}{p_t}\right)^2 \mathrm{var}(p_t)\end{aligned} \tag{A.32}$$

This leads to:

$$\mathrm{var}(\epsilon_P) = \mathrm{var}(R) + (Q\kappa)^2\,\mathrm{var}(Q) + \frac{1}{4}Q^4\,\mathrm{var}(\kappa) \tag{A.33}$$

$$\begin{aligned}\mathrm{var}(z_P) = {} & \mathrm{var}(z) + \left[\frac{p_z}{p_t}(1 - R\kappa)\right]^2 \mathrm{var}(Q) + \left(\frac{p_z}{p_t}Q\kappa\right)^2 \mathrm{var}(R) + \left(\frac{p_z}{p_t}QR\right)^2 \mathrm{var}(\kappa) + \\ & + \left(\frac{Q}{p_t}(1 - R\kappa)\right)^2 \mathrm{var}(p_z) + \left(\frac{Q}{p_z p_t^2}(1 - R\kappa)\right)^2 \mathrm{var}(p_t)\end{aligned} \tag{A.34}$$

CHAPTER B

Tools

Man must shape his tools lest they shape him.

A. R. Miller

A few novel software tools for debugging and analysis of the vertex reconstruction algorithms were written within the scope of this thesis. This chapter is dedicated entirely to these tools.

B.1 Algorithm tuning

All vertex finders have one or more parameters that can be tuned. The process of tuning can formally be described as an optimization of an objective function; in the b-tagging scenario this objective function is likely to depend on a b-tagging efficiency and a fake rate or something similar. Such an objective function basically describes the relative importance we assign to the two values. The objective function that we tune against can also be a function of the "geometric" Monte Carlo description. In this case the objective function is a measure for the congruence of the reconstructed objects with the simulated objects. The advantage of such a geometry based objective function as opposed to a b-tagging based one is that it is not specific to a physics case. It does not involve b-tagging code, it is more direct. The results are easier to understand and interpret. The advantage of the b-tagging based objective function is that it is specific to a physics case. Physicists will find results based on a b-tagging related objective function more interesting. Understanding an algorithm from the results of tuning in b-tagging, on the other hand, is incomparably more complicated.

This section will walk through the steps of implementing a package that automates the tuning of an algorithm against an arbitrary objective function. The current implementation realizes only an optimization against a "geometric" objective function; a few possible generalizations of the package are discussed in the last subsection (Sec. B.1.6).

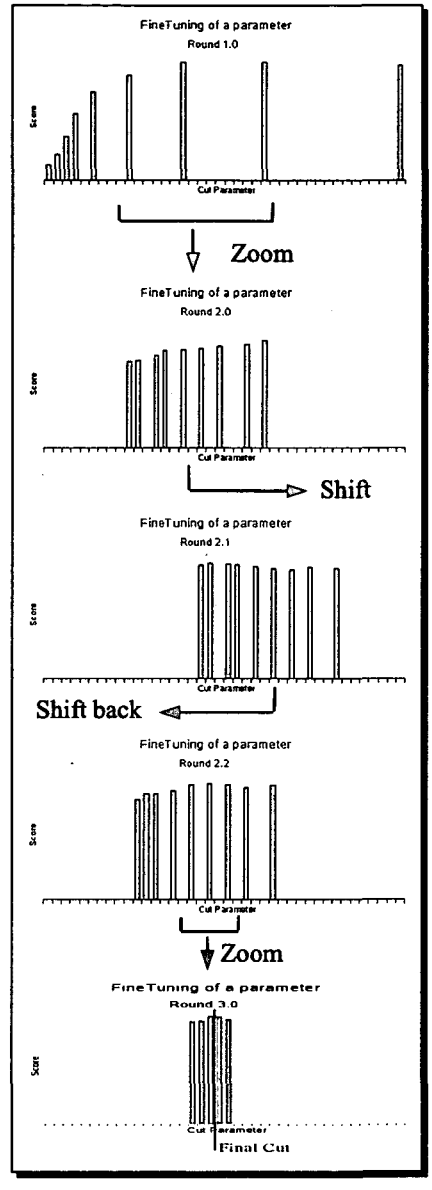


FIGURE B.1:

B.1.1 An MC based "geometric" score

For historical reasons, the general objective function that operates on geometric properties only, is called the vertex finding score or simply the "score". It is a measure of congruence between the simulated event and the reconstructed event. It is implemented in the class VertexFindingScore, its definition reads:

$$S = E_{\mathrm{Pr}}^{a} \cdot E_{\mathrm{Sc}}^{b} \cdot P_{\mathrm{Pr}}^{c} \cdot P_{\mathrm{Sc}}^{d} \cdot A_{\mathrm{Pr}}^{e} \cdot A_{\mathrm{Sc}}^{f} \cdot (1-F)^{g} \qquad \text{(B.1)}$$

Here E denotes the "finding efficiency", P is short for "vertex purity", A means "assignment efficiency", and F stands for "fake rate"; "Pr" denotes primary vertices, "Sc"

secondary vertices. The parameters a, b, c, d, e, f, g are SimpleConfigurables - they can be changed at runtime. The defaults are $a, b, c, d, e, f, g = 2, 2, .5, .5, .5, .5, 1$.

Note that the score heavily depends on the description of the simulated event; whether or not a decay vertex appears in the description as a separate SimVertex changes heavily the final score. It is not always trivial what SimVertices should appear in the event description. Clearly the decay vertex of a Higgs boson should not appear – it is under any circumstances indistinguishable from a primary vertex. Whether or not e.g. a D meson that is a decay product of a B meson should appear as a separate SimVertex depends on the use case. To this end some manipulators that meddle with the Monte Carlo truth have been written (see subsection B.1.2).

B.1.2 Changing the MC truth

It has already been established in the previous subsections, as well as in the chapter on performance comparisons (Ch. 6) that classes that change the Monte Carlo truth are important for a quite a spectrum of algorithmic analyses. The ultimate goal must be to have a set of well-understood MC truth meddlers which establish strong correlations between geometric and e.g. b-tagging performances. This goal implies that the important features of a specific use case (e.g. b-tagging) are well understood. And only with such an understanding of the problem will it be possible to understand the merits and problems of the different vertex reconstruction strategies at a profound level. Work on such tools has barely started in ORCA. It will still take a lot of effort to reach these ambitious goals.

B.1.3 A b-tagging score

The current implementation of the tuning framework does not yet have the facilities to tune against b-tagging performance; but certainly the objective function would have to resemble:

$$S = \sum_i \delta_{b_i,t_i} \tag{B.2}$$

if i denotes the jet i, and

$$b_i = \begin{cases} 1 & \text{if } b_i \text{ is a b jet,} \\ 0 & \text{otherwise} \end{cases} \tag{B.3}$$

$$t_i = \begin{cases} 1 & \text{if } t_i \text{ is tagged,} \\ 0 & \text{otherwise} \end{cases} \tag{B.4}$$

An implementation of this score for ORCA would be desirable.

B.1.4 The TuningTools package

The TuningTools package is designed to be a framework that allows the fine tuning of various parameters against specific event data. Its design should accommodate for a maximum of

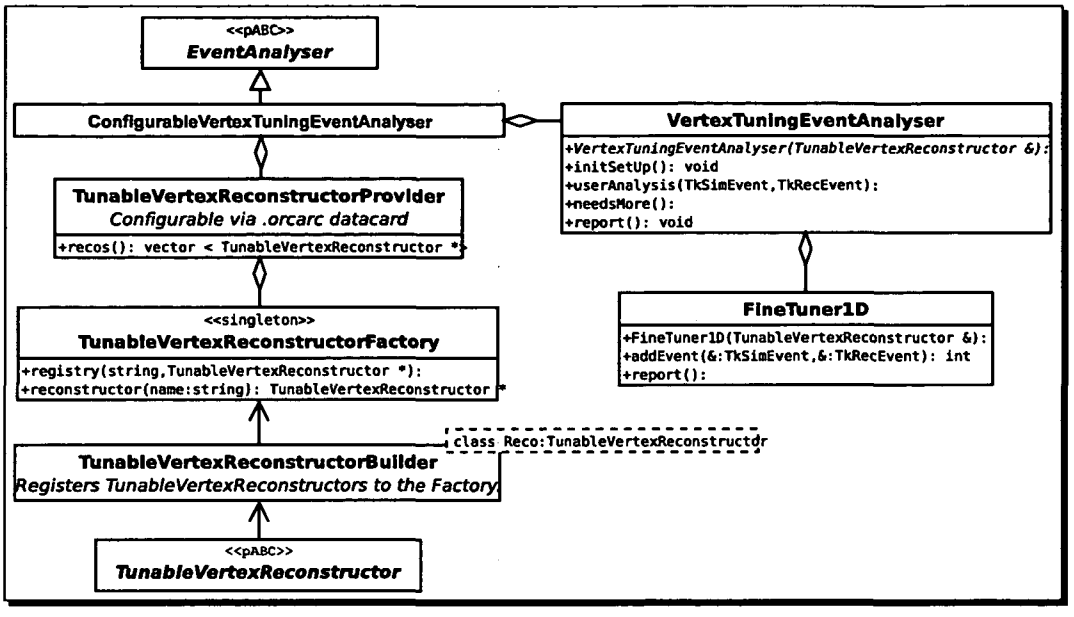


FIGURE B.2: UML class diagram of the FineTuning framework.

flexibility; the e.g. optimizing algorithm and the objective function should be exchangeable. Any reconstructor should certainly be tunable against any of its parameters.

The current implementation does not implement the first two of the aforementioned design goals; both the optimizing algorithm and the objective function are currently hard coded. Future developments would have to address these two constraining issues.

Implementation details

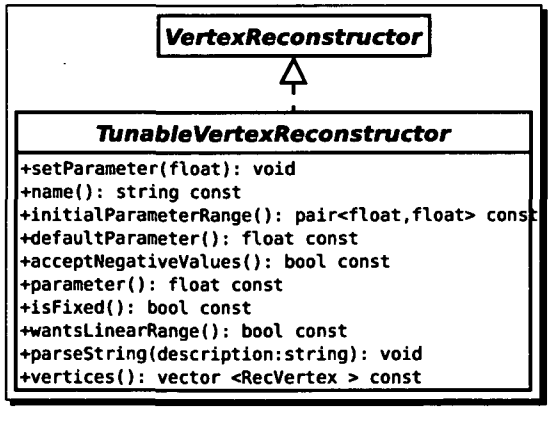


FIGURE B.3: TunableVertexReconstructor interface

The most important abstract base class is the *TunableVertexReconstructor* (Fig. B.3). The user implements a *TunableVertexReconstructor* for every parameter in every VertexReconstructor that he/she wants to see tuned. With this class the user primarily specifies how the parameter is tuned, but also a range of where the optimal parameter value can be expected to be found. The user also specifies whether the range should be "scanned" logarithmically or linearly. Additionally it defines a good default value for an algorithm. This makes it possible to compare novel algorithms with a well-defined base line.

As it can be seen already from the interface, a lot of issues in this package are written with a one-dimensional parameter scan in mind. A generalization will trigger many changes throughout the package.

The TuningTools package employs an abstract factory TunableVertexReconstructorFactory to cleanly separate implementation of the *TunableVertexReconstructors* from the tuning tools framework, see Fig. B.2. The *TunableVertexReconstructors* self-register to the factory. The class with the impossible name ConfigurableVertexTuningEventAnalyser serves as the glue code between CARF and TuningTools. From an orcarc configurable it determines what algorithms the user wants to try, queries the TunableVertexReconstructorProvider for the instantiations of the *TunableVertexReconstructors* and sets up the VertexTuningEventAnalyser. Later on, when G3EventProxy data is dispatched, the ConfigurableVertexTuningEventAnalyser dispatches the data further to its set of VertexTuningEventAnalysers. Note that the mechanism is flexible: via the orcarc configurable the user can set various algorithmic-specific parameters, add fixed-parameter algorithms to the list (to e.g. compare algorithms against a baseline), or change the initial parameter range. All these features are implementable within the concrete *TunableVertexReconstructor* classes.

B.1.5 The FineTuner1D

The FineTuner1D class (see Fig. B.4) implements a 1d optimization technique by "scanning" a certain parameter range, then deciding upon where the maximum "score" can be expected (see Fig. B.1). Depending on where within the interval the maximum is found, the algorithm either "zooms in", or shifts the parameter range and repeats the procedure. This is done a few times, until, in a final round, a set of fine-tuned algorithms are compared.

The user can configure many aspects of the procedure:

- The number of events per "round",
- the number of events in the "final round",
- the initial parameter range,
- the number of "zooms" that are performed (this is referred to as the "depth"),
- the number of bins that cover the parameter ranges at every iteration of the tuning process,
- whether or not the parameter range should be binned linearly or logarithmically, and

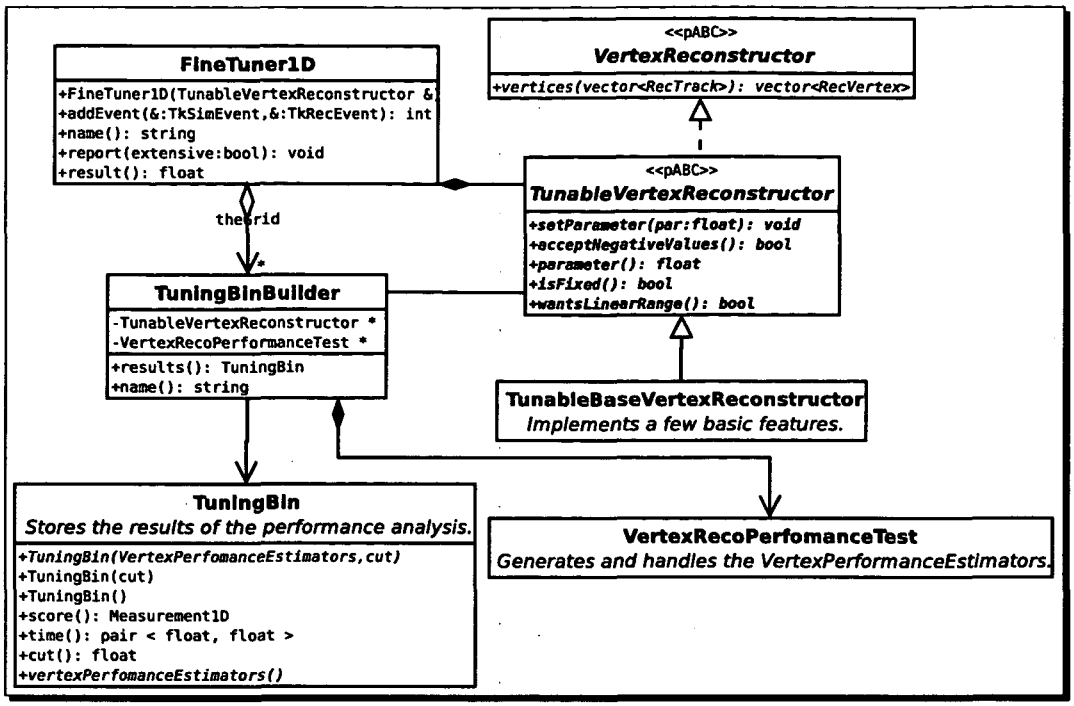


FIGURE B.4: UML class diagram of the 1D tuning algorithm.

- the verbosity of the procedure.

Details of the FineTuner1D

The FineTuner1D (Fig. B.4 internally uses TuningBinBuilder. One such builder has a specific VertexReconstructor (with a specific value of the tunable parameter) and a specific VertexPerformanceTest instantiation. FineTuner1D dispatches the input events to the TuningBinBuilders. It is the TuningBinBuilders task to call the VertexReconstructor, and to analyse and store the result. The stored result is a TuningBin. According to the results stored in the TuningBins, FineTuner1D determines its winner, and the new range. In the end, when a final "winner" is found, the final result is presented.

B.1.6 Future development

Clearly the package needs to be transformed into a more general-purpose tool. The top item on the TODO list is the abstraction of the objective function. It must be possible to optimize against non-geometric, physics-oriented, information. Maybe a second point is the abstraction of the optimization technique. Whether this is a reasonable goal is very hard to say before it is actually tried. But it is safe to say that at this stage one should consider using general optimization packages like Minuit [10]. In this case the TuningTools

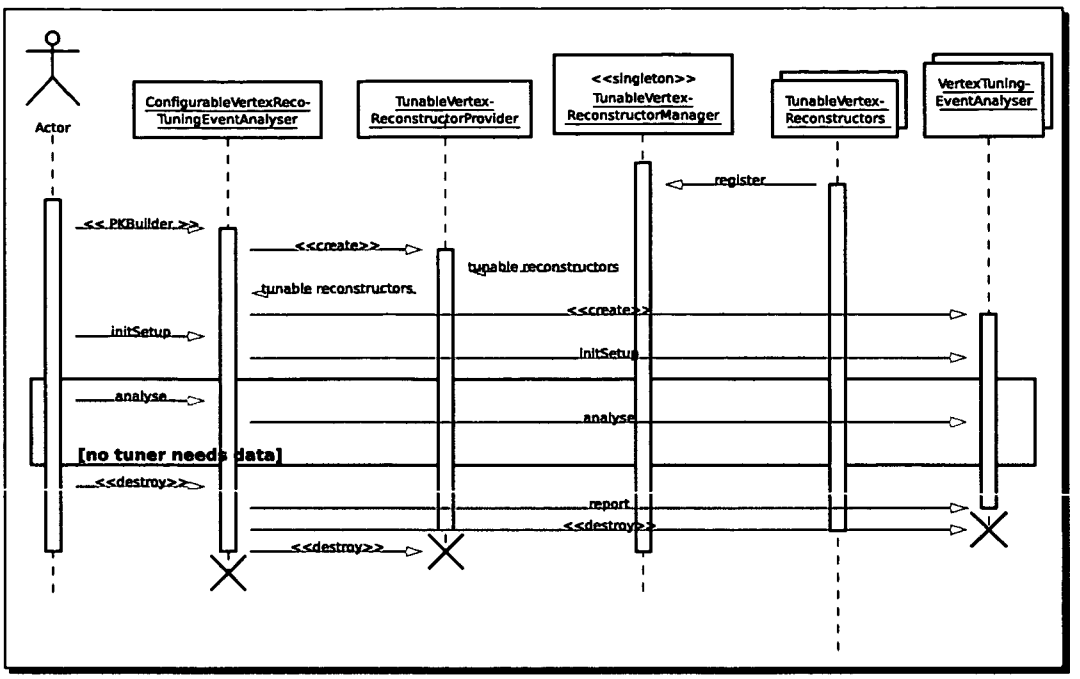


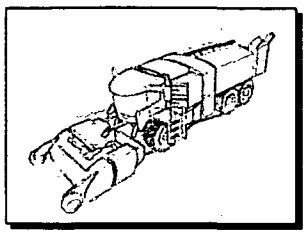
FIGURE B.5: Sequence diagram of the tuning tools.

would turn into a much smaller package that only interfaces to a general minimization package.

B.2 Data harvesting

Say not ye, There are yet four months, and then cometh harvest? behold, I say unto you, Lift up your eyes, and look on the fields; for they are white[1] *already to harvest.*

The New Testament, John 4:35



One frequent task common to both debugging and analyzing algorithms is the collection of data in ntuples. These days ROOT [29] seems to be the most often used tool for such tasks in the high energy physics community. What the user has to do in ROOT to produce data tuples and/or histograms, is the following:

- Instantiate a ROOT file,
- Instantiate the ntuple objects,
- fix the format of the ntuples, i.e. name the columns, define the data type of the columns, in case of arrays, fix the size of the arrays (or name the variable that defines the size of the array). Also define the variables that will be used for filling the ntuples.
- Every time the user wants to fill an ntuple row, he/she has to
  - Copy the data into the variables defined above,
  - Call ntuple::Fill()
- Finally, write and close all ntuples.

This lengthy procedure clearly carries the burden of FORTRAN legacy — the fact that the user has to fill certain variable "blocks" is reminiscent of the old common blocks. For a modern programming environment it is very insecure and inflexible. (The "fill" variables, the TFile, and the TTrees have to be known wherever information needs to be made persistent. A simple change in the ntuple layout triggers a change in at least two places in the source code. No checks are made against wrong variable types, variables being in the wrong order, etc.) In the 21st century things ought to be done differently.

The data harvesting concept has been introduced as a remedy against many of these problems. It intends to fulfill the following design goals:

- **Simple tasks must be simple** — only a few simple lines of code should have to be written for storing e.g. two floats in an ntuple.
- **Complicated tasks must be achievable** — very "convoluted" use cases need not be simple, but they should still be coped with.

[1] ripe

- **User code should be technology agnostic** — changing from storing data into ROOT files to storing into AIDA [19] files should not imply significant changes in the user code.[2]

- **The user shall have to handle as few objects himself as possible** — it is not necessary for the user in these use cases to be exposed to e.g. file handles. It must be enough for him/her to state "I want to store data X into file named Y". The same argument holds for ntuple objects.

- **The user shall not have to supply redundant information** — If the user supplies only floats for a column of an ntuple, he/she evidently wants the column type to be a float. He/she should not have to specify this explicitly; implicit specification is enough, simpler, and more robust. The worst design clearly is to let the user specify types explicitly, and then not check whether the user supplied data is consistent with the specification. This is how the ROOT API currently works. In the case that the user supplies different types for one column, the more general type should be used. If this is not possible, a warning or an error message should be issued.

- **Changes in what is stored, e.g. adding another column in an ntuple, must be an easy task** — it should trigger changes in only one place in the code. This is especially important for debugging tasks, where the content of an ntuple is likely to change frequently.

- **It should be as difficult as possible for the user to introduce inconsistencies** — separating the definition of the ntuple and the filling of the ntuple is prone to introducing such inconsistencies.

- **It is desirable to be able to document what exactly is stored in what column in what ntuple.**

- **Everything related to getting the data into a file should happen in only one place in the code** — and that place should be close to where the data is produced.

- **Filling one ntuple from various places - e.g. in more than one source files - should be as easy as possible**

- **Producing such ntuples should be non-intrusive** — debugging an algorithm should not imply major changes in the algorithm code.

The implementation of the data harvester fulfills all of the aforementioned design goals — except for one tiny issue about closing the files, and the fact that some more complicated tasks can be realized only in a very poor manner; the shortcomings will be described later.

[2]This issue has also been addressed in AIDA via its plugin mechanism.

As the fundamental data type a MultiType is introduced that heavily exploits the C++ capability of operator overloading. The MultiType (but not only the MultiType) implements the fourth point in the list of design goals - the data type is implicitly defined with the data itself. An ntuple row is supplied either column by column (the user sequentially provides the column name and the column data; when a column name gets repeated, the object assumes the start of the next row in the ntuple), or via a map <string, MultiType>. Both constructs are very "natural" in C++ code. Both ways of supplying data are also particularly fail proof — it is almost impossible for the user to provide inconsistent data, to swap variables, or to specify the wrong data type.

The following subsections will now elaborate on the implementation of the data harvesting concept.

B.2.1 The MultiType

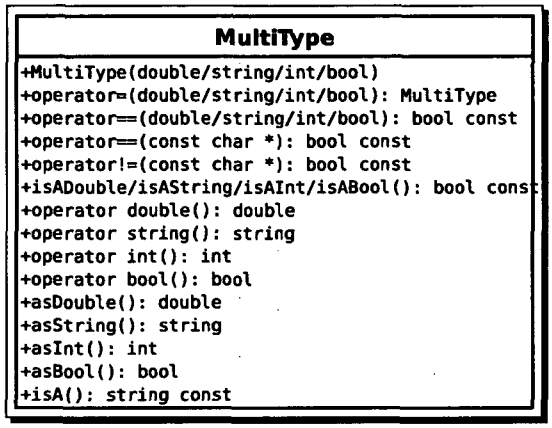


FIGURE B.6: The MultiType class.

It has already been mentioned that the fundamental data type in the DataHarvesting package is the MultiType. Fig. B.6 shows the UML description of the class. It is something of a "generalization" of four C++ types: doubles, STL strings, ints, and bools. It can be assigned any of these four types (and a few more), and it stores not only the value, but keeps also track of the data type — something of a simplistic form of real-time type information (RTTI). It also handles conversions between these types. In some sense, it softens the strongly typed paradigm of C++.

B.2.2 The AbstractDataHarvester

The *AbstractDataHarvester* has been introduced to allow for a technology agnostic data harvester, one of the design goals stated in section B.2. Implementations of this class will be specific to one technology - there will be one implementation for ROOT file formats,

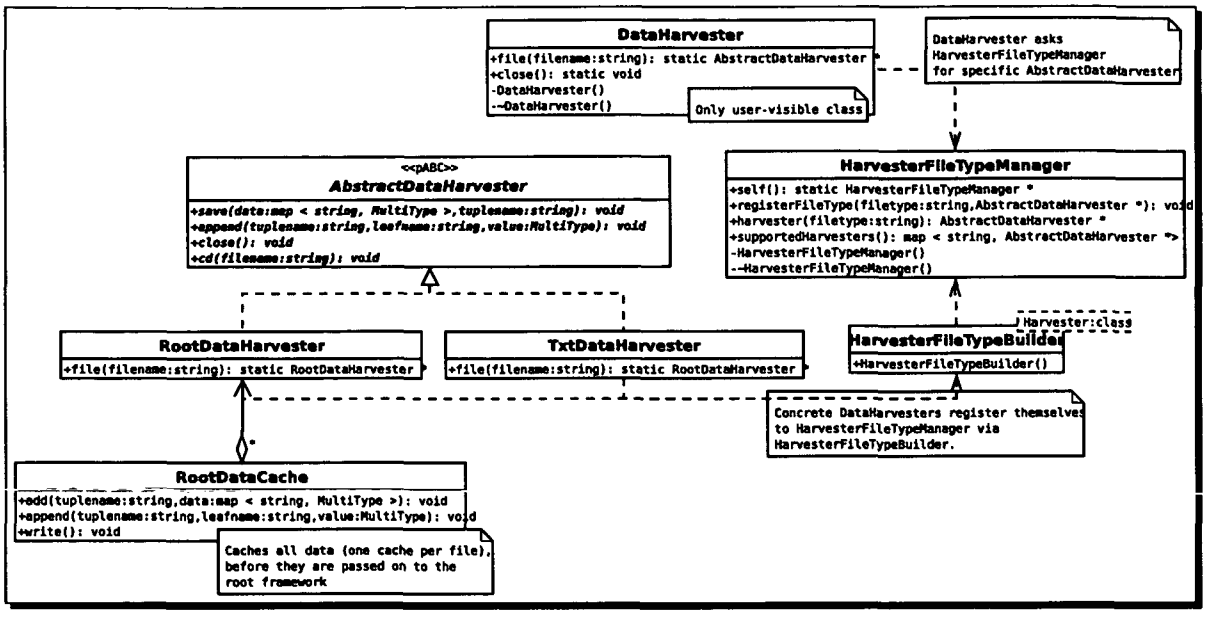


FIGURE B.7: The DataHarvesting framework

one for AIDA, etc. The purely abstract base class is depicted in B.7. It has two methods that deal with retrieving information from the user. The first one is ::save (const map < string, MultiType > & data, string tuplename), in which the user describes an ntuple called "tuplename" and already supplies one complete "data row" at the same time! If the user ever accidentally supplies data with different layout but the same tuplename, the harvesters complain and discard the data. The second way to supply data to the harvester is via: ::append(string ntuple, string parameter, MultiType value). In this case the user supplies the data leaf by leaf. Once a leaf is named again, the harvesters will assume that one data row is finished, and start the next row. Again, strict consistency checks are made.

B.2.3 Implementations of the AbstractDataHarvester

Currently there are only two implementations of the *AbstractDataHarvester* interface: the RootDataHarvester and the TxtDataHarvester. The TxtDataHarvester can stream either into a simple txt file or to stdout. The file format is very simple (and redundant), it is:

```
ntuplename:  name1=value1, name2=value2, name3=value3
ntuplename:  name1 ...
```

The RootDataHarvester (see Fig. B.8) owns RootDataCache objects (one per TFile), which in turn create RootInternalNtuple, one per ntuple (TTree). Apart from introducing a more structured design, the RootDataCache — nomen est omen! — caches the data in a special, compact form (exploiting STL techniques), before writing them into the files. The data are made persistent after explicit ::write() and ::close() method calls, or if the cache has

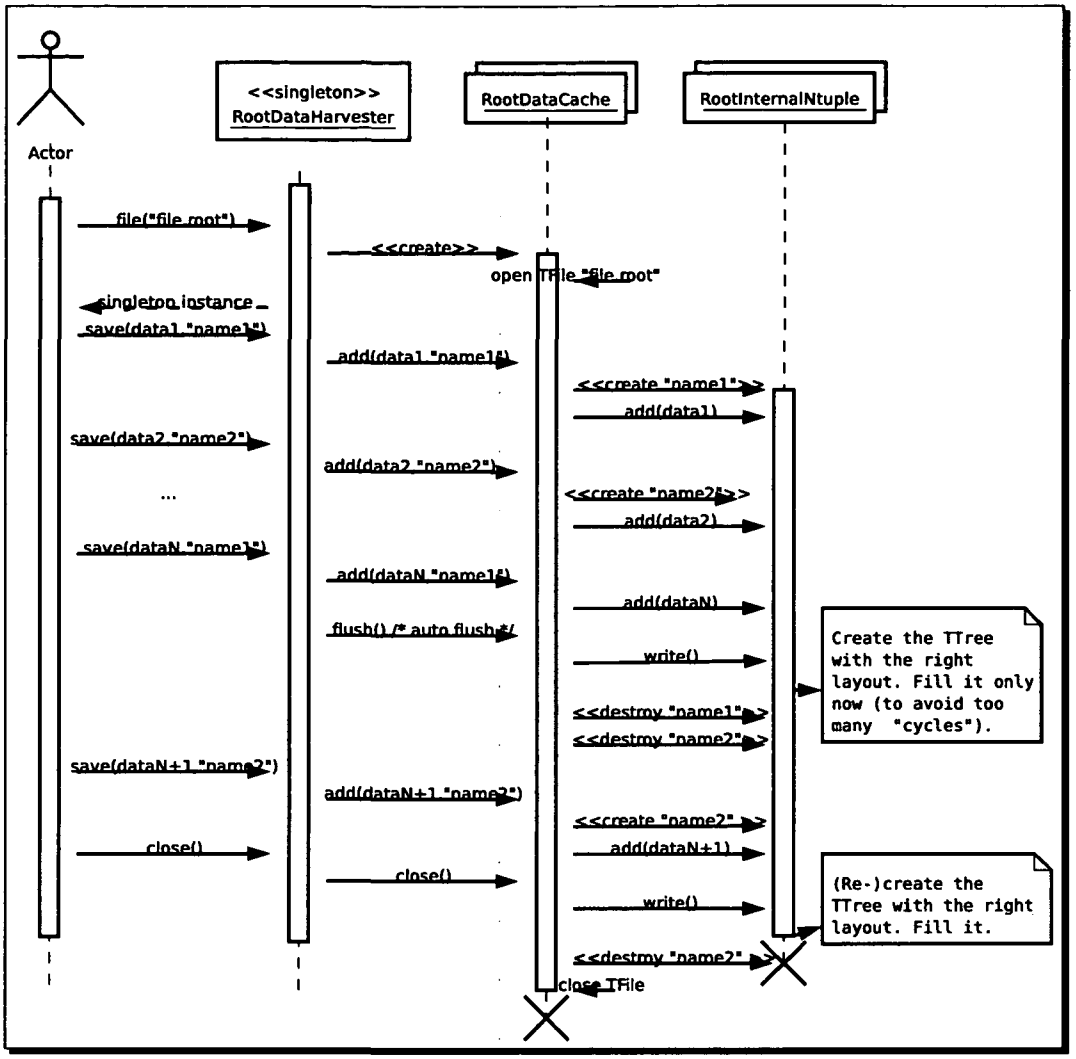


FIGURE B.8: RootDataHarvesting internals. The "actor" could in principle be user code, though in practice only the DataHarvester should be a direct "user".

exceeded its internal limit (after a certain number of saved "ntuple rows", configurable via `RootDataHarvester::FlushAfter=10000`).

ROOT files are buffered files, so why bother introduce yet another buffer? Because, with every "change" from writing into one ntuple to writing into another, ROOT introduces another "cycle" of the same ntuple. It is the author's subjective feeling that the way these cycles are handled in ROOT is painful for the user. They should thus be at least avoidable, if not avoided in the general case.

Note that the concrete harvesters are singleton classes: there is one harvester per file format, not per file. One harvester must handle all the files of a particular file format.

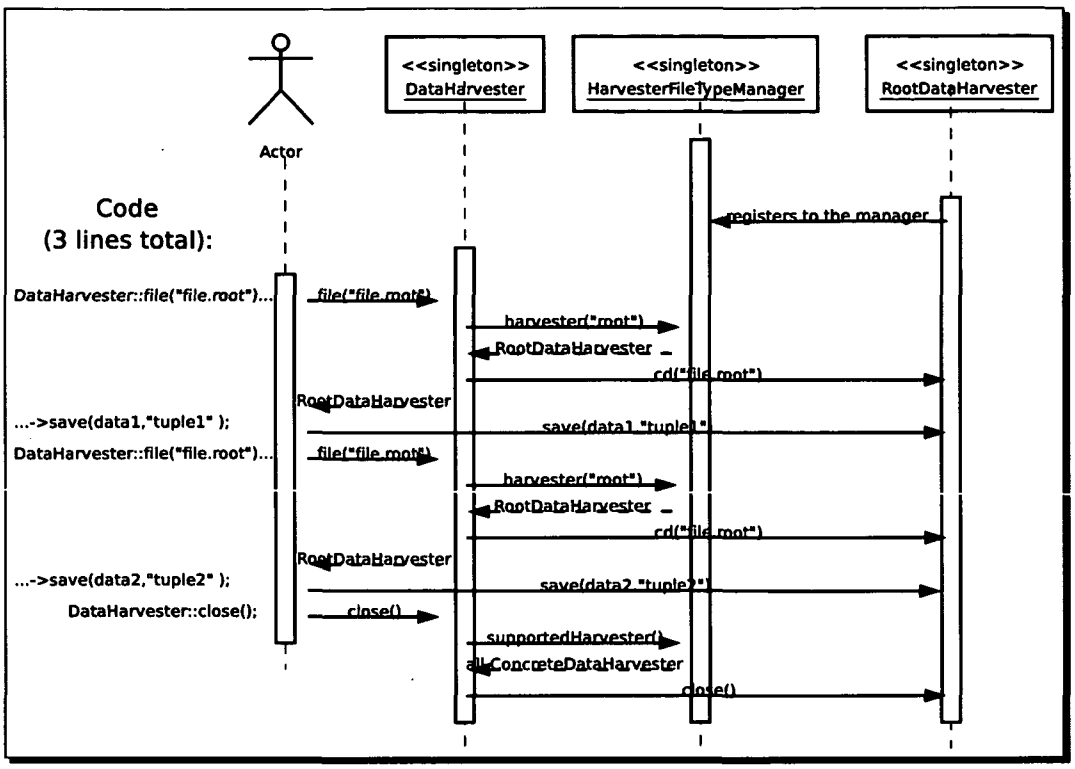


FIGURE B.9: Data harvesting - a sequence diagram.

It is up to the harvester to decide upon how to handle multiple open files. The fact that these harvesters are singletons is historically motivated — the feature of hiding specific technologies from the user (the DataHarvester, see next section), the concept of having one front-end for all different file formats, was introduced only later.

B.2.4 The DataHarvester

The DataHarvester (see Fig. B.7) can be seen as the front-end to the whole DataHarvesting package — it is the only class that the user will directly interact with. Its central method is ::file(string filename), which "dispatches" the right concrete data harvester to the user. The information of which concrete harvester is to be used for which file name, is provided by the HarvesterFileTypeManager. The concrete harvesters register themselves to the manager via the HarvesterFileTypeBuilder. The manager can be seen as an *abstract factory* — the technology dependent parts are confined to a well-defined set of classes. There is no class that has to know about all concrete harvesters at compile time. Thus no class depends on more than one technology. The different harvesters also do not have to appear in the same package. What harvester is available to a program can thus be a choice of what libraries the program is linked against. Fig. B.9 shows the sequence diagram of what happens when

the code snippet in the figure's leftmost column is executed.

B.2.5 Meta data

The DataHarvesters support meta data; every ntuple can be supplied with a description string, so can every column in an ntuple, with the description given in brackets, together with the column name — see the example in section B.2.7. This meta data is, itself, stored as ntuples (__ntuples__ has all ntuple descriptions; __columns__ lists all descriptions of ntuple columns).

B.2.6 Closing the harvester

Some technologies, e.g. ROOT, need to explicitly close the files. Otherwise the user suffers from data loss. The data harvesting framework is designed such that one and only one line of code is needed to close all open files: DataHarvester::close(). Currently the user has to call this function in his code. A future version of the DataHarvesting package might try to do this automatically, via e.g. the POSIX compliant atexit function. The big problem with this automatic call is that it is very hard to guarantee that all the persistency technologies are still available when the atexit function is called. (Surely the "closing method" should be registered as late as possible). Even if such an approach turns out to work in a specific environment, it might be simply impossible to guarantee that it will work with every version of every persistency technology.

B.2.7 Data harvesting in action

This subsection gives a brief demo of how the data harvester is used:

```
map <string , MultiType> m;
m["Where (Where was the exception caught)"] = "Right here";
m["What  (Description of the exception)"]   = myexception.what();

DataHarvester:: file("ex.root")->save( m,
    "Exceptions (This ntuple keeps track of all exceptions thrown)");
```

For a possible visualization of the data "reaped" with the simple code snippet given above, see Fig. B.10.

B.2.8 Object harvesting

The idea of "object harvesting" is to exploit the functionality of the data harvester for high level ORCA objects like a RecTrack or a RecVertex. See Fig. B.11 for an UML of a few object harvesters that have been written. As opposed to CARF's RecObj mechanism, the writer of an object harvester has to specify what data of an object gets stored. This is a price that is paid for having a lot of flexibility and complete independence from the ORCA/CARF framework. We shall exploit this feature in the visualization program.

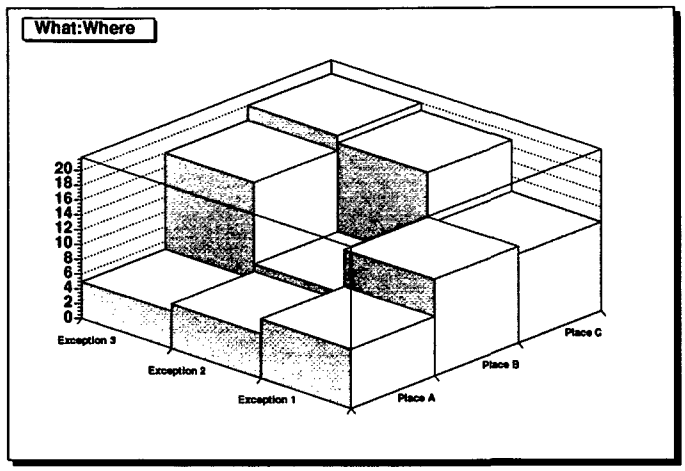


FIGURE B.10: Sample ROOT lego plot of harvested data

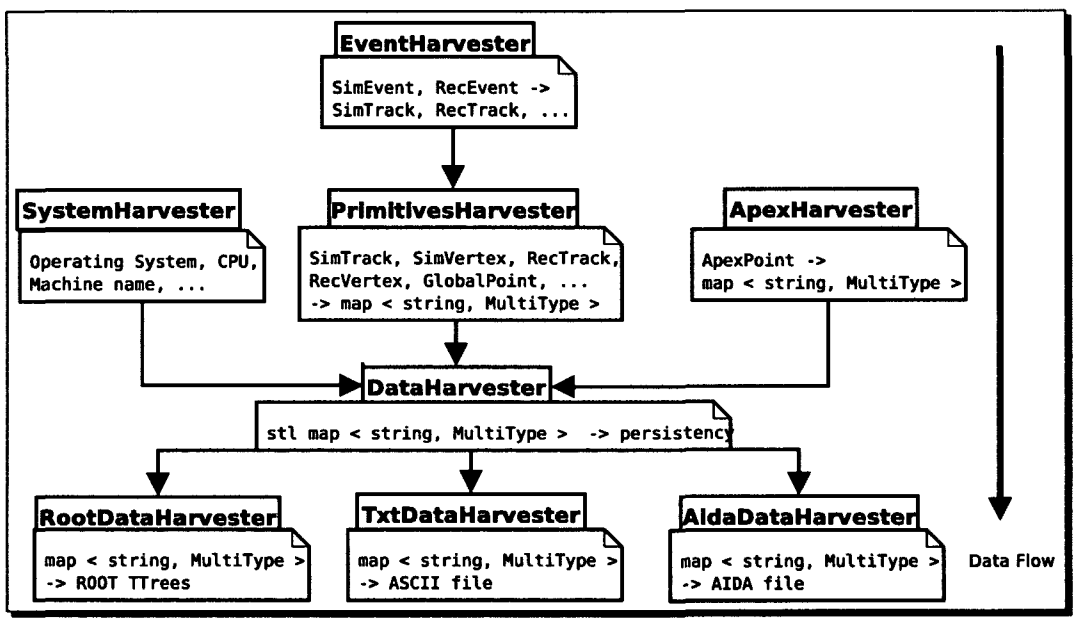


FIGURE B.11: All harvesters and their interrelationships.

```
vector < RecVertex > result;
 // Now make the RecVertices persistent. Make sure they'll
// show up green in the visualization.
map < string , MultiType > attributes;
attributes["color"]="green";
PrimitivesHarvester::file("myfile.root")->save(result, "RecVertex", attributes);
```

The sample code above shows the usage of the PrimitivesHarvester class. Using these object harvesters will produce data that can be visualized with the visualization package described in Sec. B.3.

B.2.9 Data seeding

The DataHarvester is a class that maps a map < string, MultiType > on a piece of data in a file. Clearly, also the inverse operation is defined. This operation is called a DataSeeder; it creates a number of maps < string, MultiType > from one or more files. DataSeeders are used in the e.g. visualization package (section B.3), and also — exploiting a dirty trick — in the VertexGunFromFile, see Sec. 2.2.5. Currently only a TxtDataSeeder and a first version of a "user-visible" DataSeeder have been written; a RootDataSeeder is still missing.

B.2.10 Object seeding

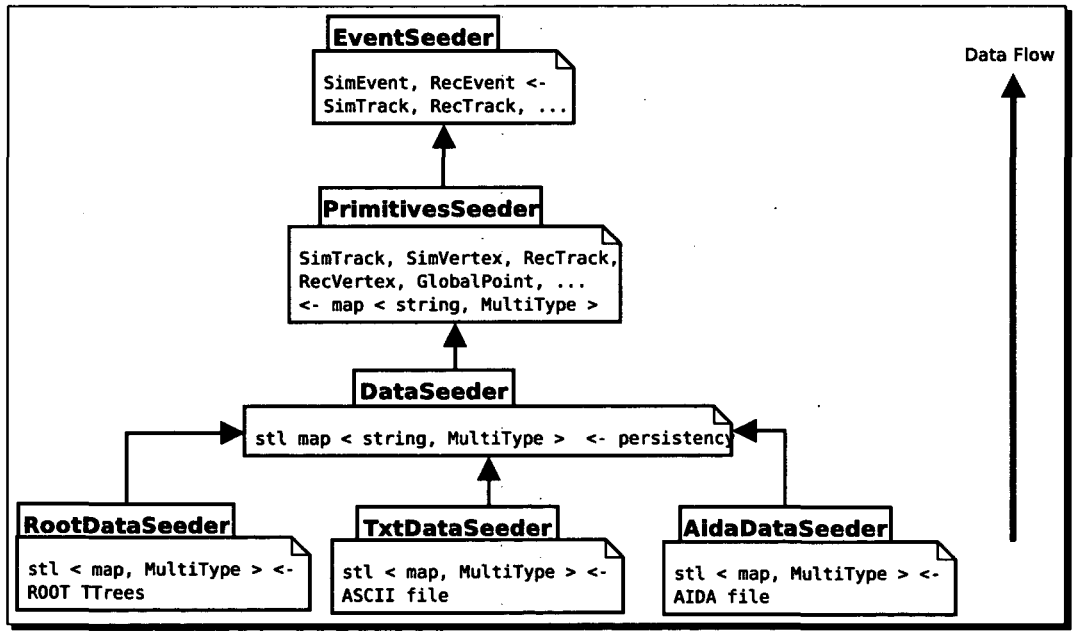


FIGURE B.12: The object seeders - the inverse operations to the object harvesters (see Fig. B.11)

The "inverse" class of a DataHarvester is called a DataSeeder. It creates an STL map < string, MultiType > from a file. No fully encapsulated object seeders exist yet. The class VertexGunFromFile (see Sec. 2.2.5) has the implementation of a PrimitivesDataSeeder. One of the next versions of the VertexFastSim package will clearly have to extract this functionality from the VertexGunFromFile and put it into a separate class.

B.2.11 Final remarks and potential future developments

Albeit it is designed to address only simple tasks, the data harvesting concept seems to have quite some potential. One major weakness of the current harvesters is that all data are stored in a "flat" way — no hierarchy in the data is (yet) foreseen. A future development will address this issue. Another shortcoming of the current implementation is its lack of any kind of arrays.

Comparing it with CARF's persistency mechanism seems somewhat nonsensical, too different use cases are addressed. Still, one nice feature of CARF's persistency mechanism is that the user never needs to specify how a specific class gets saved. The class just needs to derive from a special base class (RecObj); this alone makes objects of this class "storable".

The data harvesting concept addresses completely different needs — it tries to save (any) specific data in a very simple manner. Imitating the RecObj mechanism is not an aim of the harvesters. One should explore a true persistency solution like CARF for such tasks.

Another powerful idea is to have python pickle files [13] as an additional back-end. This could allow for a very straightforward and powerful way to analyze data, with e.g. hippodraw [7] or ROOT-python — bypassing ROOT files altogether, while keeping ROOT's supreme histogramming capabilities. Two other potential new back-ends are an optimized binary format, and the legacy HBOOK tuple format, which is still the default in e.g. the Belle collaboration.

One final idea that should be presented here, is to have data written to and read from network streams. For example, a TcpDataHarvester and a TcpDataSeeder would be a nice tool — recycling much of the TxtData(Harvester|Seeder) or the BinaryData(Harvester|Seeder) code seems reasonable. A potential use case could be "online" visualization of an event via the visualization toolkit presented in Sec. B.3. A TcpDataHarvester and a TcpDataSeeder would also be a major step towards a "harvester server". The only missing piece of code would then be just glue code that combines all the required components into a piece of server code: the user would write to a TcpDataHarvester that passes the data to the TcpDataSeeder which may reside on another machine. This simple reuse of already existing components can create the functionality that is comparable to Belle's ntuple-server ([68]). Code reuse should also make the development of converters trivial: they are nothing but a seeder and a harvester "glued" together.

B.3 Visualization

CMS has one official large visualization project: IGUANA [96]. IGUANA is currently actively developed; it is extremely feature rich. The graphics it produces are both aesthetically appealing and functional. Still, a separate special purpose visualization tool has been written in the context of this PhD thesis. The reasons for this development are manifold. One main motivation was simply "it could be done"; other reasons may seem more

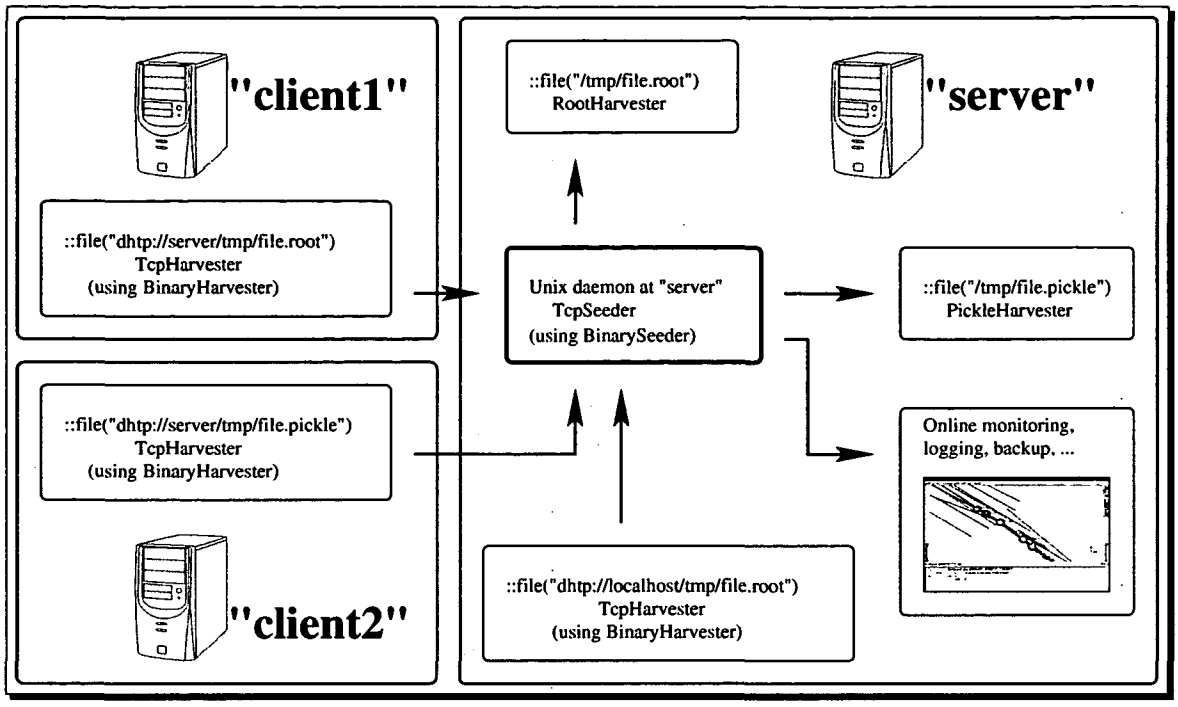


FIGURE B.13: A vision of what a future harvesting usage could look like. Three client applications talk to one central Unix daemon via the DataHarvester Transfer Protocol (dhtp). The server passes the requests to local harvesters.

convincing:

- The author wanted to have a visualization toolkit that does not depend on CERN(HEP) tools, CERN(HEP)-specific setups, etc. A toolkit that depends only on standard tools is easier to setup, maintain, and port.
- A special purpose tool, evidently, focuses on the task at hand. It can be tailored better towards the use case.
- For our specific context it can be much simpler and faster than IGUANA.
- There is no user base that needs to be served (at least not yet). This directly translates into maximum freedom for the code writer.

The concept itself is implemented in two well separated parts. One part consists of the object harvesting classes. As has been mentioned earlier, many of the objects saved by the object harvesters can be visualized. This part of the visualization code resides within ORCA. The other part is the visualization package itself. It is completely disjoint from ORCA. In fact it needs no HEP-specific tool at all. Data reading is done in a **DataSeeder** implementation that "ships" with the rest of the visualization code. For

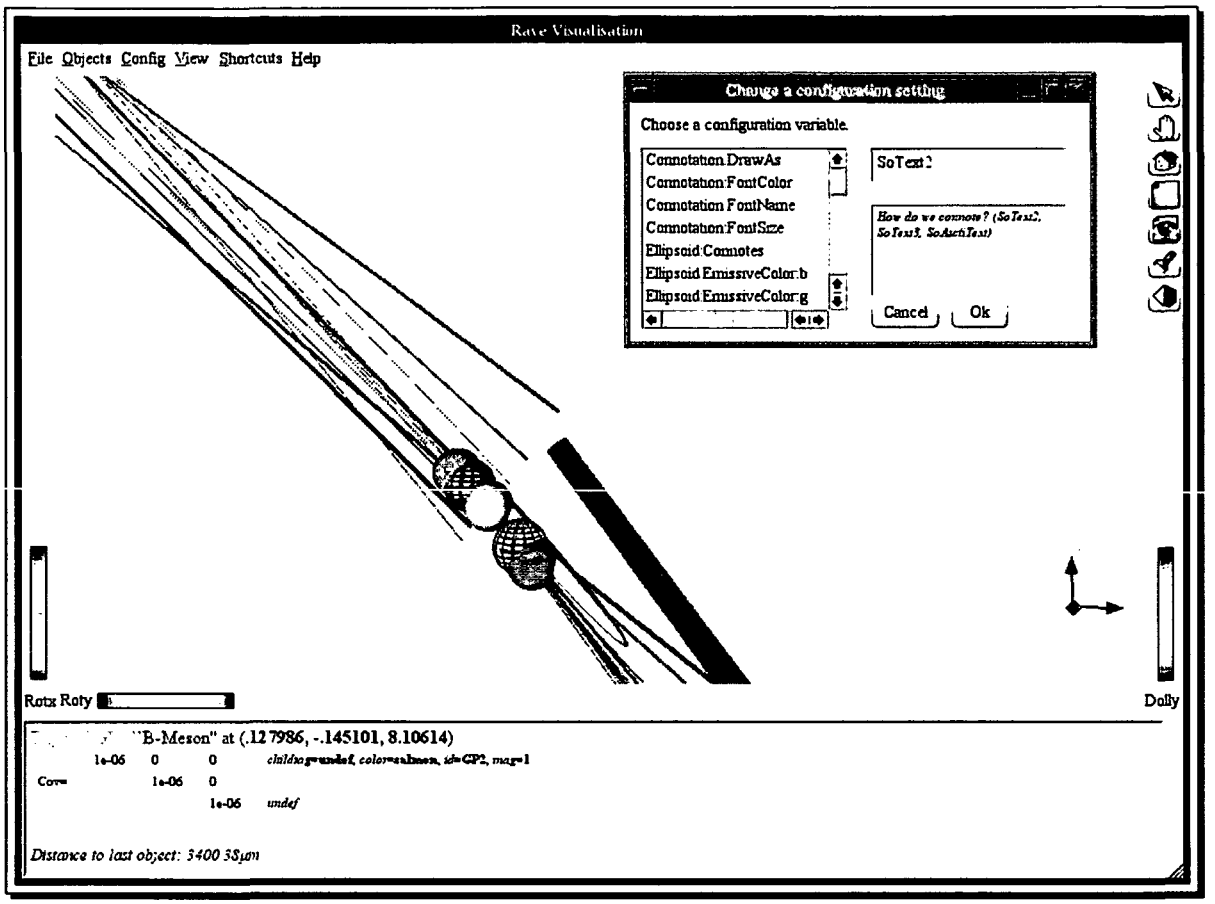


FIGURE B.14: Screenshot of a VertexSimpleVis visualization of a $b\bar{b}$ event; two B mesons - 3.4 millimeter apart from one another - are discernible on the plot (magenta spheres), one of which is selected and described in the text window.

graphics, Coin3D [4] — an OpenGL-based technology — is used. The coin3d viewer is embedded in and surrounded by Qt [15] widgets. For a screenshot see Fig. B.14.

The event plots throughout this thesis are results of this tool, unless stated otherwise.

B.3.1 The offline program

It is not the aim of this section to give a complete description of the offline visualization program. Rather it shall give an overview of the main design features.

(a) **The visualization is closely tied to the Coin3d library.** This is a deficiency rather than an asset, although it is a very convenient one for the programmer.

(b) **All objects are data harvester maps: (map<string, MultiType>).** Data are consistent by convention only, not by design. Apart from a few evident drawbacks, this has a large number of pros.

(c) **All aspects of all drawable objects can be changed at runtime.** This is done via a *Manipulator* class (see Fig. B.15) — and by virtue of (b) one manipulator class is enough for all objects!

(d) **As many global aspects as possible should be configurable at runtime.** Fig. B.14 shows the "configurable changer". The configurables, too, are map<string, MultiType>.

(e) **Configuration options, view, etc. must be "storeable".** They are put in a file in a DataHarvester-compatible form.

(f) **One must be able to save manipulated event data to a file.** This feature is still missing.

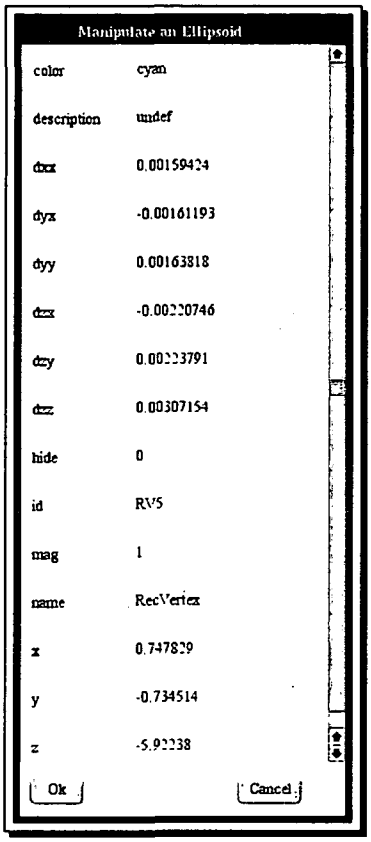


FIGURE B.15: Manipulator

B.3.2 Download

The offline code can be downloaded as a tarball at:
`http://wwwhephy.oeaw.ac.at/u3w/w/walten/www/dist/vis.tar.gz`

B.4 Vertex-specific *b*-tagging analysis tools

A vertex-specific package for *b*-tagging analysis tools has been introduced only recently to ORCA by the author. The code base derives from a few more heavy-weight analysis classes of the *b*-tagging group. It has since its creation evolved into a package that is better suited towards the needs of the vertexing group. As the persistency backend ROOT has been abandoned in favor of the data harvesting tools (see Sec. B.2). The BTaggingHarvester has been introduced as the package's central class. which takes a pair < BTagInputObject,

BTagOutput > as its input and produces a few tuples from the information that is contained in the STL pair. All b-tagging related results shown in Ch. 6 are produced with this package.

B.5 Associators

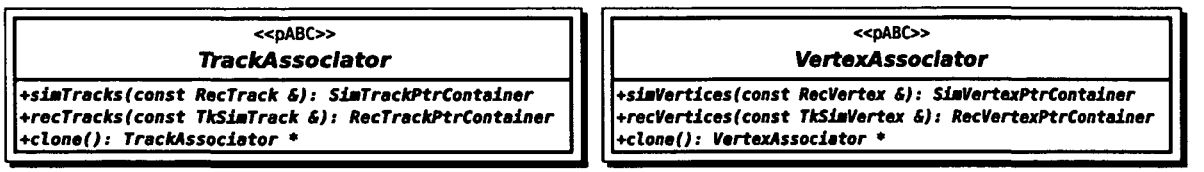


TABLE B.1: The track and the vertex associator interfaces.

In order to be able to perform studies that compare reconstructed objects with Monte Carlo truth, "links" between reconstructed objects and the simulation objects are needed. Classes that return such links are called **Associators** in CMS lingo. In the process of this PhD thesis a few minor contributions to track association and vertex association have been made, some of which shall be presented here.

B.5.1 Track association

Track association defines the task of associating **TkSimTracks** with **RecTracks** and vice versa. ORCA currently sees two implementations: one that associates by hits (**TrackAssociatorByHits**) and one that associates by a distance criterion between **TkSimTracks** and **RecTracks** (**TrackAssociatorByPulls**). Association by pulls requires a distance measure. Two classes have been implemented: **TrivialTrackDistance** simply compares the absolute value of track momenta. **TrackDistanceByChiSquared** tries to propagate the two tracks onto one common surface. It then computes a χ^2 criterion (with 5 degrees of freedom) between the tracks.

Clearly, association by hits can be expected to produce the most meaningful results. It should be used wherever possible. Real life, though, has a few arguments against an exclusive usage:

- It cannot be used with the **VertexFastSim** (Sec. 2.2.2) package, since this fast simulation package produces no hits.
- Persistent **RecTracks** in ORCA > 7_6_0 are stored neither with their hits nor with an association.

Hence, an associator that operates by pulls and performs as reliably as the association-by-hits is desirable. One cannot assume that association by hits serves as an absolute baseline. There may well be cases in which an associator-by-pulls returns a more meaningful result

than the by-hits algorithm. This makes a quantitative comparison difficult. Still, in the context of this thesis two comparisons were conceived. At first, we tried a track-by-track comparison between a by-pull associator and the standard by-hit associator. For every SimTrack it was checked, whether or not the two associators agreed on a RecTrack. Another test plots the number of RecTracks associated with each TkSimTrack. If one assumes that the track reconstructor produces reasonable results, then the deviations from 1:1 associations can be seen as indications of weaknesses of the associator. The test was performed against a $J/\psi\ \phi \rightarrow K^+K^-\mu^+\mu^-$ decay. Only the (highly collimated) SimTracks of the decay were considered (4 per event), but all the RecTracks of the whole signal event were used ($\approx 10-30$ per event). 2000 events (= 8000 tracks) were considered. Table B.2 shows the test results. TrackDistanceByChiSquared with a high cutoff ($\chi^2_{\text{cut}} \approx 5000-10000$) performs reasonably well. Note that fortunately the cutoff value is not "critical" — over a range of two magnitudes (1000 – 100000) the results vary only by one percent. Finally it shall be repeated that for VertexFastSim-generated events, a "perfect" track associator is available: TrackAssociatorByMap, see Sec. 2.2.3.

Name	(%)	0	1	2	3	4	5	6	7	8	9
ByHits	100	672	7328								
$\chi^2 < 50$	89.7	1420	6580								
$\chi^2 < 200$	93	1150	6850								
$\chi^2 < 1000$	96.2	885	7056	58	1						
$\chi^2 < 2000$	97.1	809	7014	176	1						
$\chi^2 < 5000$	97.5	757	6660	570	12	1					
$\chi^2 < 10000$	97.4	705	6077	1188	28	2					
$\chi^2 < 20000$	97.3	635	5281	2018	55	11					
$\chi^2 < 100000$	96.3	482	3694	2369	1341	92	22				
$\delta(m) < 4\%$	87.3	633	4992	1815	448	97	12	3			
$\delta(m) < 7\%$	87.7	422	4098	2194	896	277	89	19	4	1	
$\delta(m) < 15\%$	85.9	210	2101	2239	1495	961	533	267	126	52	16

TABLE B.2: Results of the track association tests for $J/\psi\ \phi \rightarrow K^+K^-\mu^+\mu^-$ tracks. The column marked as "(%)" denotes the congruence with the by-hit associator. The other columns show how often n RecTracks were associated with one SimTrack. "$\chi^2 < \ldots$" rows mark a TrackDistanceByChiSquared measure, "$\delta(m)$" denotes the use of TrivialTrackDistance.

B.5.2 Vertex association

A VertexAssociator has to be able to associate RecVertices with SimVertices and vice versa. We can divide the set of VertexAssociators into two categories: association by tracks, and association by distance. ORCA currently has one class per category. VertexAssociatorByTrack associates according to the associations between the SimVertices' SimTracks and the

RecVertices' RecTracks. It relies on a **TrackAssociator**. **VertexAssociatorByDistance** uses a distance measure to determine the compatibility between the vertices. ORCA owns a class that measures the weighted distance in 3d, and one that measures in 2d. Both can be used for this associator.

CHAPTER C

Plots and Tables

This chapter contains all plots that are considered too lengthy, too technical, and/or too detailed for the main chapters.

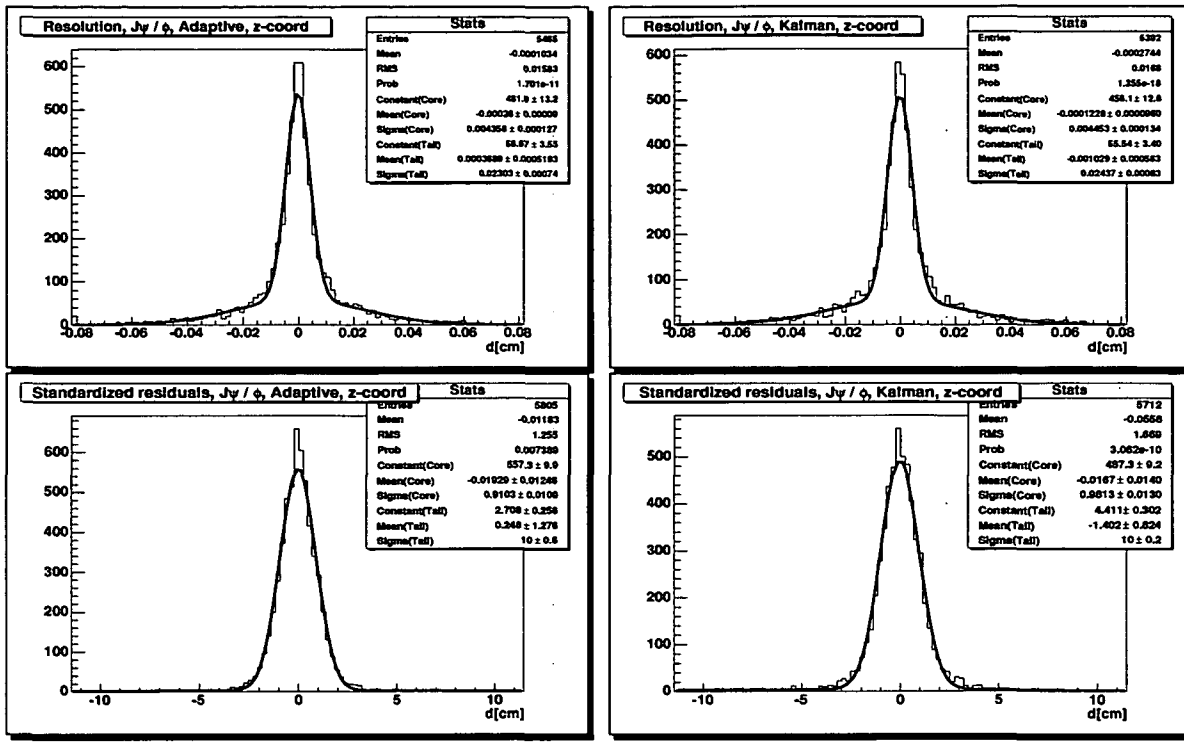


FIGURE C.1: Comparing AVF with KVF, J/ψ ϕ channel.

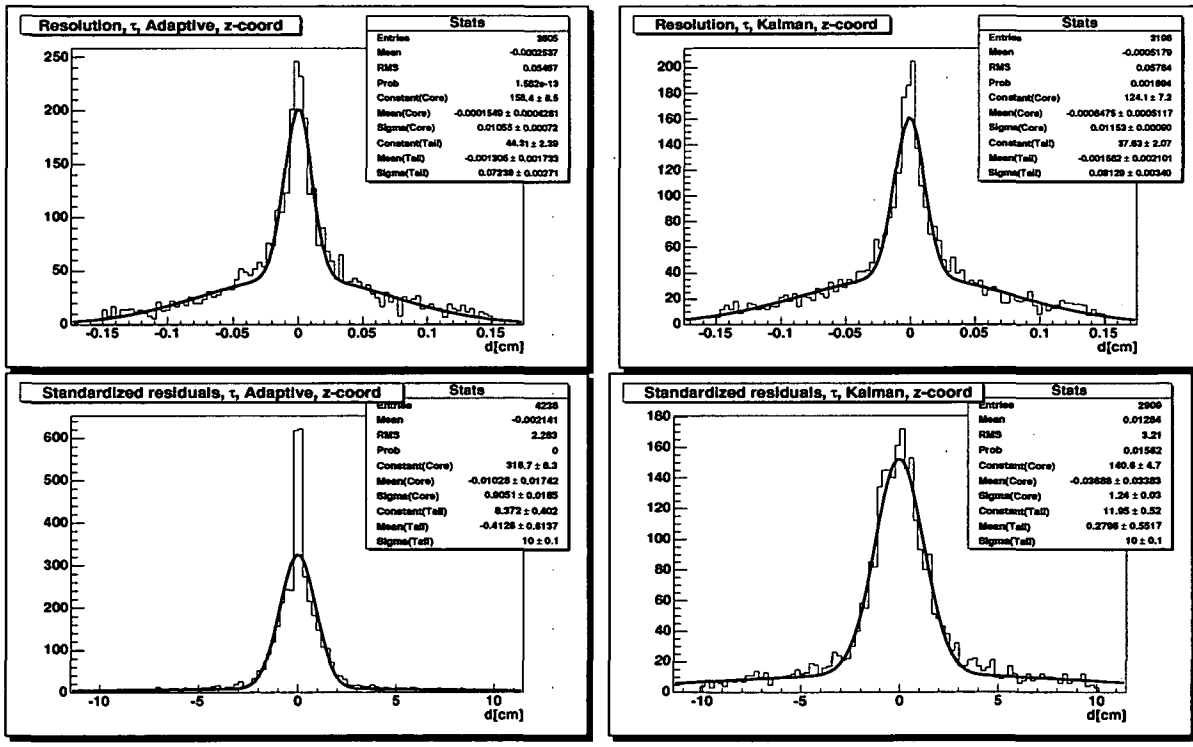


FIGURE C.2: Comparing AVF with KVF, τ channel.

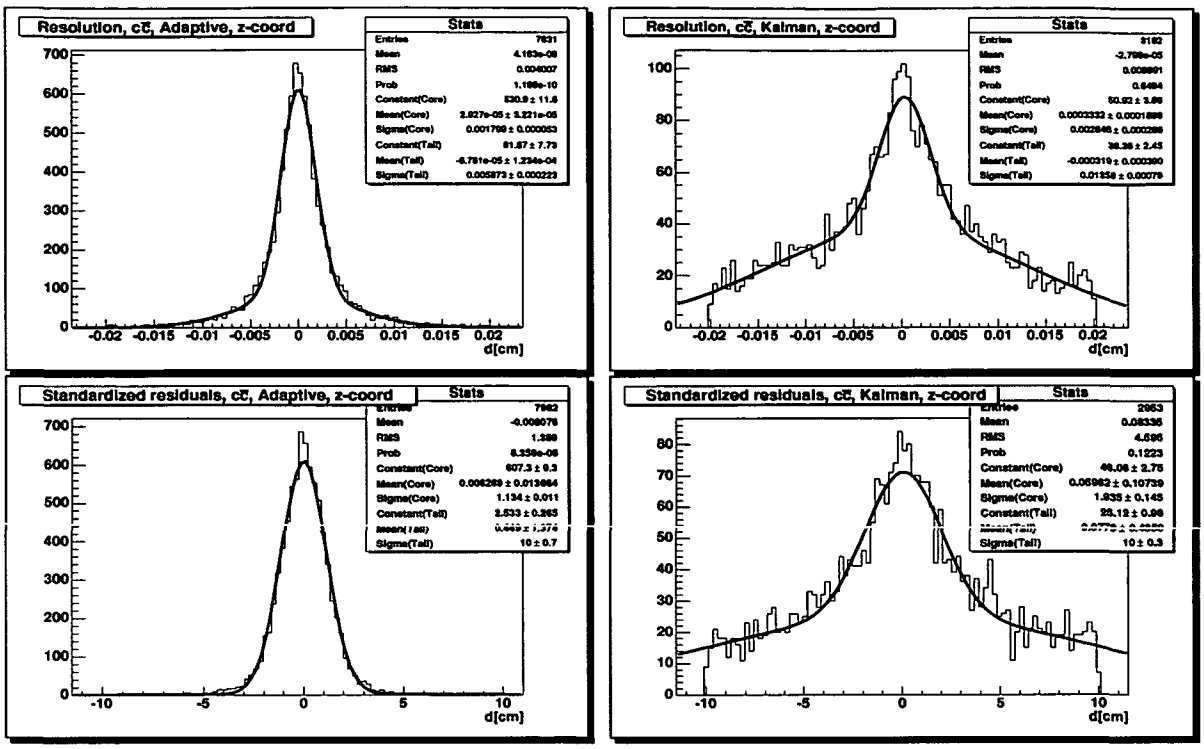


FIGURE C.3: Comparing AVF with KVF, $c\bar{c}$ channel.

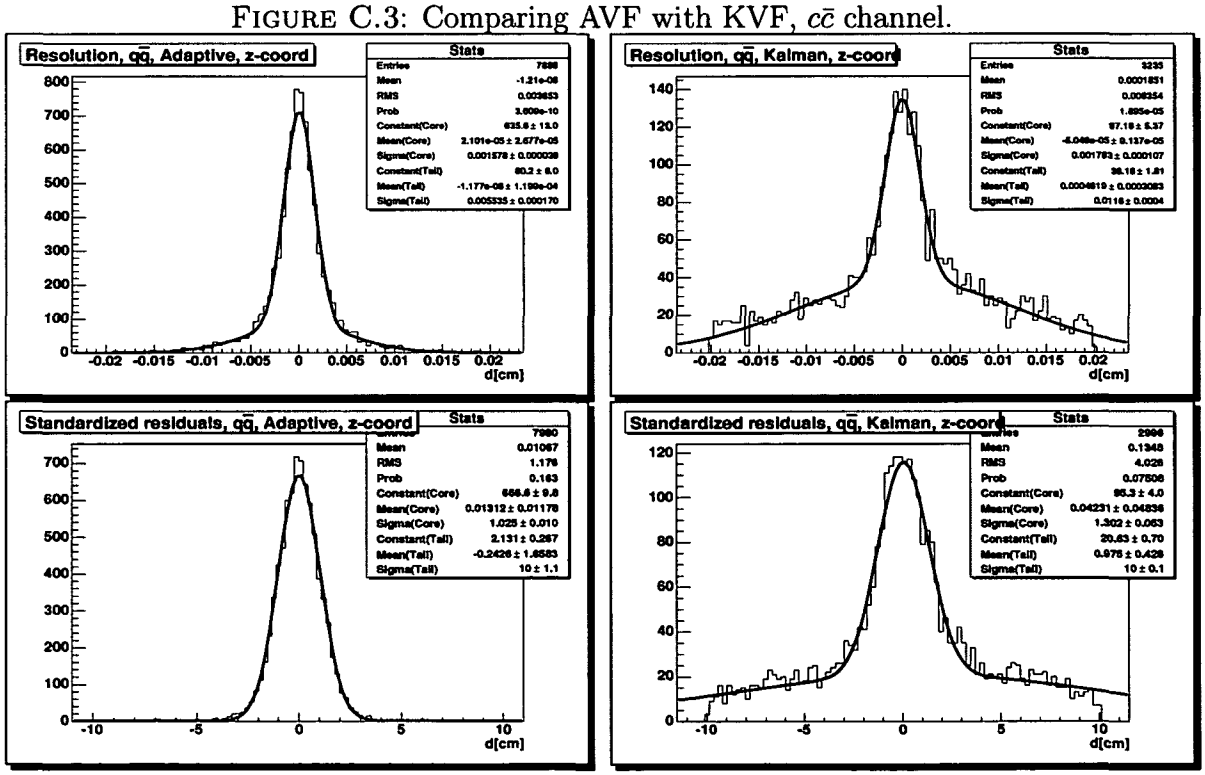


FIGURE C.4: Comparing AVF with KVF, $q\bar{q}$ channel.

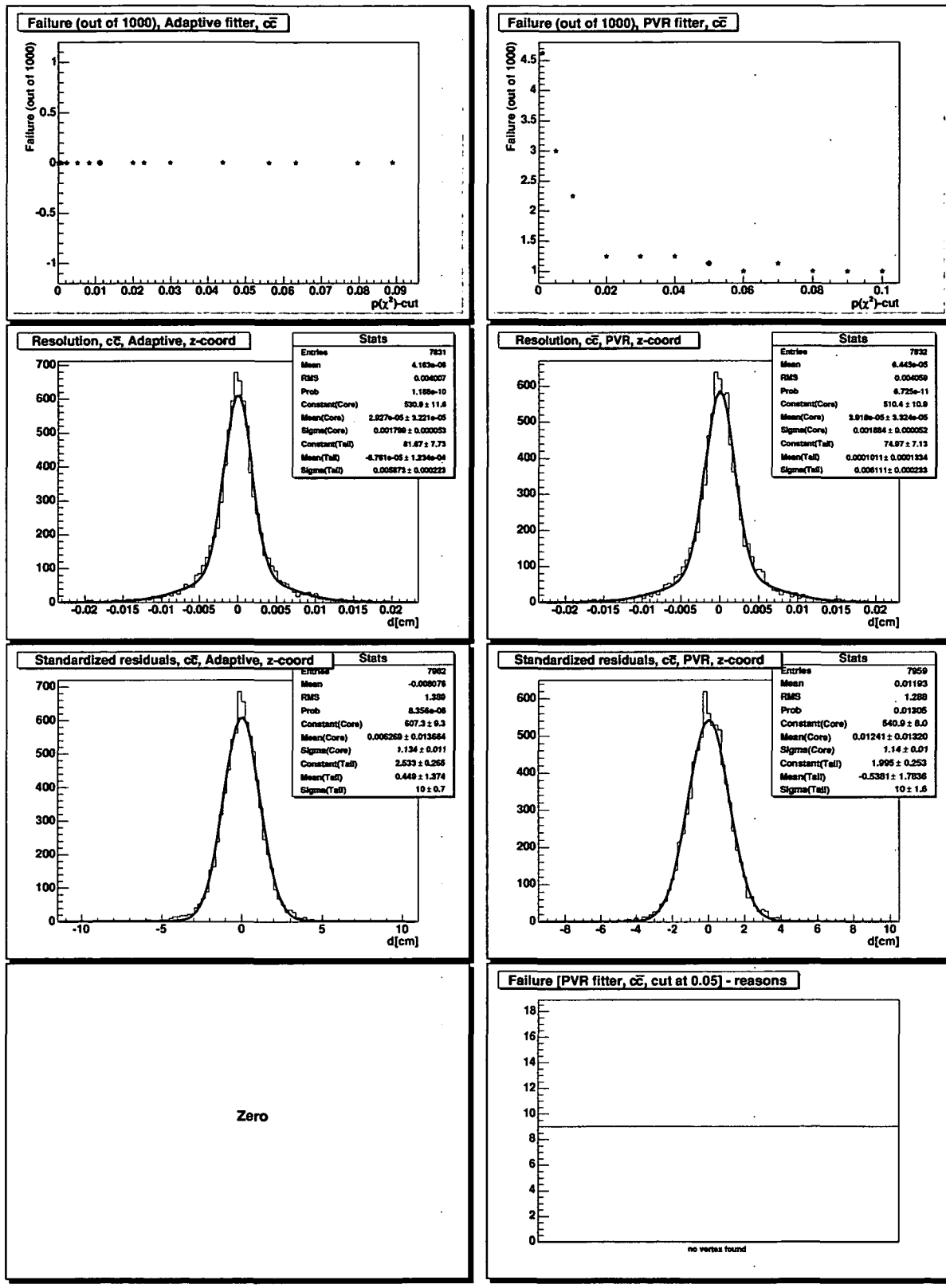


FIGURE C.5: Comparing AVF with PVF, $c\bar{c}$ channel.

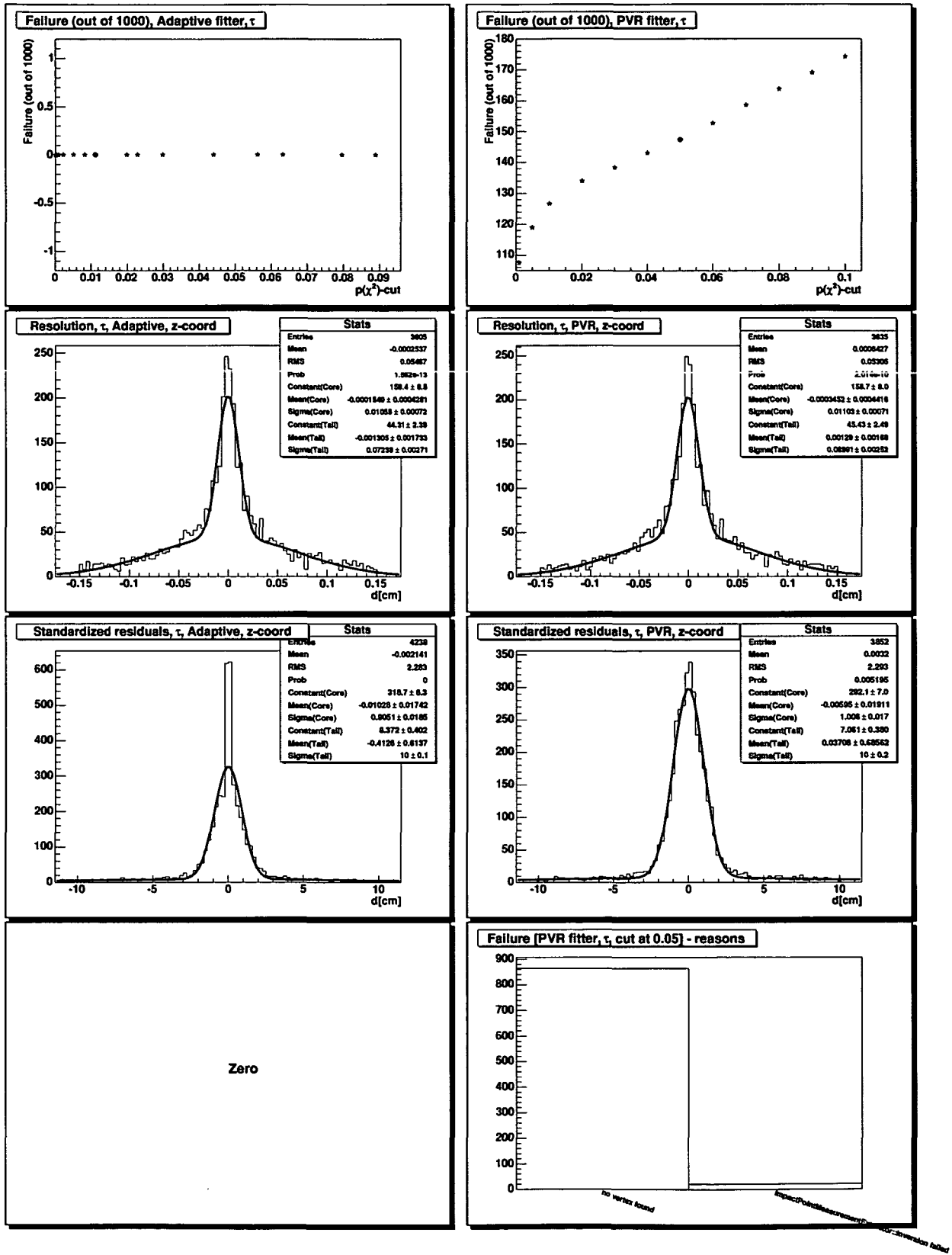


FIGURE C.6: Comparing AVF with PVF, τ channel.

LinPtFinder	$c\bar{c}$				$q\bar{q}$			
	RMS [μm]	σ_{Fit} [μm]	fail ‰	t [ms]	RMS [μm]	σ_{Fit} [μm]	fail ‰	t [ms]
SubsetHSM(-1)	38	29	0	3.5	37	25	0	4
HSM(-1)	37	27	0	3.4	32	24	0	3.9
LMS(-1)	196	41	2	3.4	237	44	11	3.9
SubsetHSM(100)	61	37	0	0.8	41	31	2	0.8
HSM(100)	54	38	0	0.7	48	32	0	0.8
AdaptiveApex(Weighted,Mtv)	565	35	202	63.8	477	34	239	71.7
AdaptiveApex(Unweighted,Mtv)	691	793	556	60.2	724	858	568	67.8
TrimmingApex(Weighted,Mtv)	148	35	21	56.1	135	32	28	63.8
TrimmingApex(Unweighted,Mtv)	147	33	16	51.3	202	37	29	59.5
LinearApex(Unweighted,Mtv)	1101	1145	446	51	1128	1275	512	58.1
Fsmw(exp=0 f=0.5 c=10 np=-1)	38	27	0	17.6	33	24	0	25.5
Fsmw(exp=-0.5 f=0.5 c=10 np=-1)	38	27	0	17.6	33	24	0	27
Fsmw(exp=-1 f=0.5 c=10 np=-1)	38	27	0	17.8	33	24	0	25.4
Fsmw(exp=0 f=0.3 c=10 np=-1)	38	28	0	14.4	33	24	0	20.1
Fsmw(exp=-0.5 f=0.3 c=10 np=-1)	38	28	0	14.4	33	24	0	20.1
Fsmw(exp=-1 f=0.3 c=10 np=-1)	38	28	0	14.4	33	24	0	20.1
Fsmw(exp=0 f=0.6 c=10 np=-1)	40	27	0	18.9	35	24	0	27.5
Fsmw(exp=-0.5 f=0.6 c=10 np=-1)	40	27	0	18.9	35	24	0	27.5
Fsmw(exp=-1 f=0.6 c=10 np=-1)	40	27	0	18.9	35	24	0	27.5
Fsmw(exp=0 f=0.4 c=10 np=-1)	38	27	0	16.1	34	24	0	22.9
Fsmw(exp=-0.5 f=0.4 c=10 np=-1)	38	27	0	16.1	34	24	0	22.9
Fsmw(exp=-1 f=0.4 c=10 np=-1)	38	27	0	16.1	34	24	0	22.9
Fsmw(exp=-2 f=0.5 c=10 np=-1)	38	27	0	17.6	33	24	0	25.4
Fsmw(exp=-2 f=0.4 c=10 np=-1)	38	27	0	16.1	34	24	0	22.9
Fsmw(exp=-2 f=0.3 c=10 np=-1)	38	28	0	14.4	33	24	0	20.1
Fsmw(exp=-2 f=0.6 c=10 np=-1)	40	27	0	18.9	35	24	0	27.5
Fsmw(exp=-2 f=0.5 c=50 np=-1)	38	27	0	17.6	33	24	0	25.4
Fsmw(exp=-1 f=0.5 c=50 np=-1)	38	27	0	17.6	33	24	0	25.4
Fsmw(exp=-2 f=0.5 c=20 np=-1)	38	27	0	17.6	33	24	0	25.3
Fsmw(exp=-2 f=0.5 c=30 np=-1)	38	27	0	17.6	33	24	0	25.4
SMS(-1)	89	28	0	121.7	141	28	3	207.8
SMS(100)	127	32	1	2.1	166	31	6	2.1
ISMS(-1)	29	23	0	159	35	22	0	270.1
ISMS(100)	56	31	0	2.7	33	28	1	2.7
FixedSMS(-1)	233	46	2	121.7	239	41	15	207.7
FixedSMS(100)	265	53	10	2.1	288	49	16	2.2
ISMS(200)	33	27	0	8.2	30	26	1	8.1
Default	49	33	0	3	43	29	0	3

TABLE C.1: Resolutions and failure rates of different LinearizationPointFinders for quark pairs.

LinPtFinder	$J/\psi\ \phi \to K^+K^-\mu^+\mu^-$				$\tau^\pm \to \pi^\pm\pi^+\pi^-$			
	RMS [μm]	σ_{Fit} [μm]	fail ‰	t [ms]	RMS [μm]	σ_{Fit} [μm]	fail ‰	t [ms]
SubsetHSM(-1)	744	61	194	0.2	1018	780	379	0.1
HSM(-1)	744	61	194	0.2	1020	780	378	0.1
LMS(-1)	833	139	170	0.2	1035	937	384	0.1
SubsetHSM(100)	744	61	194	0.2	1018	780	379	0.1
HSM(100)	744	61	194	0.2	1020	780	378	0.1
AdaptiveApex(Weighted,Mtv)	615	59	220	1.9	895	643	354	1.8
AdaptiveApex(Unweighted,Mtv)	821	323	277	1.8	868	560	404	1.6
TrimmingApex(Weighted,Mtv)	645	65	213	1.3	900	593	346	1.1
TrimmingApex(Unweighted,Mtv)	781	179	260	0.8	876	553	329	0.7
LinearApex(Unweighted,Mtv)	761	267	240	0.7	924	642	369	0.7
Fsmw(exp=0 f=0.5 c=10 np=-1)	757	66	183	0.3	1028	878	377	0.2
Fsmw(exp=-0.5 f=0.5 c=10 np=-1)	657	68	147	0.3	1017	822	366	0.2
Fsmw(exp=-1 f=0.5 c=10 np=-1)	614	69	142	0.3	993	769	349	0.2
Fsmw(exp=0 f=0.3 c=10 np=-1)	742	61	197	0.3	1027	681	376	0.2
Fsmw(exp=-0.5 f=0.3 c=10 np=-1)	668	61	164	0.3	1022	787	371	0.2
Fsmw(exp=-1 f=0.3 c=10 np=-1)	606	64	147	0.3	1000	725	352	0.2
Fsmw(exp=0 f=0.6 c=10 np=-1)	753	65	185	0.3	1034	864	375	0.2
Fsmw(exp=-0.5 f=0.6 c=10 np=-1)	646	66	150	0.3	1023	849	364	0.2
Fsmw(exp=-1 f=0.6 c=10 np=-1)	608	66	144	0.3	1002	791	345	0.2
Fsmw(exp=0 f=0.4 c=10 np=-1)	744	61	194	0.3	1019	780	379	0.2
Fsmw(exp=-0.5 f=0.4 c=10 np=-1)	670	61	166	0.3	1019	801	372	0.2
Fsmw(exp=-1 f=0.4 c=10 np=-1)	618	65	143	0.3	993	761	355	0.2
Fsmw(exp=-2 f=0.5 c=10 np=-1)	625	70	156	0.3	970	774	323	0.2
Fsmw(exp=-2 f=0.4 c=10 np=-1)	578	63	140	0.3	967	755	323	0.2
Fsmw(exp=-2 f=0.3 c=10 np=-1)	569	63	142	0.3	977	729	325	0.2
Fsmw(exp=-2 f=0.6 c=10 np=-1)	620	68	158	0.3	977	794	319	0.2
Fsmw(exp=-2 f=0.5 c=50 np=-1)	599	68	134	0.3	983	797	331	0.2
Fsmw(exp=-1 f=0.5 c=50 np=-1)	625	65	140	0.3	1009	789	356	0.2
Fsmw(exp=-2 f=0.5 c=20 np=-1)	606	70	148	0.3	970	768	322	0.2
Fsmw(exp=-2 f=0.5 c=30 np=-1)	598	70	143	0.3	979	785	323	0.2
SMS(-1)	740	62	162	0.5	1022	877	379	0.1
SMS(100)	740	62	162	0.2	1022	877	379	0.1
ISMS(-1)	736	60	162	0.6	1010	876	382	0.2
ISMS(100)	736	60	162	0.2	1010	876	382	0.2
FixedSMS(-1)	711	63	173	0.5	1018	830	384	0.1
FixedSMS(100)	711	62	173	0.2	1018	830	384	0.1
ISMS(200)	736	60	162	0.4	1010	876	382	0.2
Default	578	64	140	0.2	967	755	323	0.1

TABLE C.2: Resolutions and failure rates of different LinearizationPointFinders for low-multiplicity jets. See the text for detailed description.

CHAPTER D

Acronyms

TLAs are a PIA.

source unknown

AIDA Abstract Interfaces for Data Analysis [19]

API Application Program Interface

ATLAS A Toroidal LHC ApparatuS [1]

AVF AdaptiveVertexFitter

AVR AdaptiveVertexReconstructor

CARF CMS Analysis and Reconstruction Framework [67]

CLHEP a Class Library for High Energy Physics [2]

CAOH Closest Approach On Helices

CMS Compact Muon Solenoid

COBRA Coherent Object-oriented Base for Reconstruction, Analysis and simulation [66]

CMSIM CMS Simulation [33]

DAF Deterministic annealing filter

DAQ Data Acquisition

DST Data Summary Tape

ECAL Electromagnetic Calorimeter

FAMOS FAst MOnte carlo Simulation

FSMW Fraction-of Sample Mode with Weights

GAC Global Association Criterion

GSF Gaussian Sum Filter

GUI Graphical User Interface

GUT Grand Unified Theory

HCAL Hadronic Calorimeter

HSM Half Sample Mode

HLT High Level Trigger

HEP High Energy Physics

IGUANA Interactive Graphics For User ANAlysis [96]

KM K-Means

KVF KalmanVertexFitter

LCG LHC Computing Grid [9]

LEP Large Electron Positron collider

LHC Large Hadron Collider

LMS Least Median of Squares

LS Least (sum of) Squares

LVF LinearVertexFitter

MC Monte Carlo

MSSM Minimal SuperSymmetric Model

mSUGRA minimal SUperGRAvity

MVF MultiVertexFitter

MVR MultiVertexReconstructor

NDF Number of degrees of freedom

LTS Least Trimmed Sum of Squares

ORCA Object Oriented Reconstruction for CMS Analysis [38]

OSCAR Object oriented Simulation for CMS Analysis and Reconstruction [37]

PAW Physics Analysis Workstation [12]

PIA Pain In the Ass

PtCA Point of Closest Approach

PVR PrincipalVertexReconstructor

PVF PrincipalVertexFitter

RMS Root Mean Square

QFT Quantum Field Theory

SMS Small Median of Squares

SPC Super-paramagnetic clustering

STL Standard template library

SUSY Supersymmetry

SVF SequentialVertexFitter

TLA Three Letter Acronym

TTMD TwoTrackMinimumDistance

UML Unified Modeling Language [17]

VRML Virtual Reality Markup Language

VQ Vector Quantization

Index

Bibliography

[1] ATLAS, A Toroidal LHC ApparatuS. http://atlas.web.cern.ch/Atlas/.

[2] CLHEP — A Class Library for High Energy Physics. http://wwwinfo.cern.ch/asd/lhc++/clhep.

[3] CMSIM Events Page. http://cmsdoc.cern.ch/cmsim/pictures/cmsim_events.html.

[4] Coin3d. http://www.coin3d.org.

[5] The DataGrid Project. http://cern.ch/eu-datagrid.

[6] HEPVIS. http://www-pat.fnal.gov/graphics/HEPVis/www.

[7] Hippodraw. http://www.slac.stanford.edu/grp/ek/hippodraw/.

[8] The Large Hadron Collider. http://lhc-new-homepage.web.cern.ch/lhc-new-homepage/.

[9] LHC Computing Grid Project. http://lcg.web.cern.ch/LCG/Overview.htm.

[10] The MINUIT software package. http://seal.web.cern.ch/seal/snapshot/work-packages/mathlibs/minuit/index.html.

[11] ORCA's vertex subsystem. http://cmsdoc.cern.ch/cms/Physics/btau/management/activities/reconstruction/vertex/vertex.html.

[12] PAW – Physics Analysis Workstation. http://wwwasd.web.cern.ch/wwwasd/paw/.

[13] Pickle – Python object serialization. http://docs.python.org/lib/module-pickle.html.

[14] POOL, Pool Of persistent Objects for LHC. http://lcgapp.cern.ch/project/persist/.

[15] Qt. http://www.trolltech.com/products/qt/index.html.

[16] SEAL, Core Libraries and Services Project. http://seal.web.cern.ch/seal/.

[17] UML, Universal Modeling Language. http://www.uml.org.

[18] K. Abe et al. Time Dependent B_s^0–$\overline{B_s^0}$ Mixing Using Inclusive and Semileptonic B Decays at SLD. *Nuclear Instruments and Methods in Physics Research A*, 446:53–58, 2000.

[19] Academic Software Organization. AIDA, Abstract Interfaces for Data Analysis. http://wwwasd.web.cern.ch/wwwasd/lhc++/AIDA/.

[20] W. Adam, R. Frühwirth, A. Strandlie, and T. Todorov. Reconstruction of Electron Tracks with the Gaussian-Sum Filter. CMS-RN-2003-001.

[21] W. Adam, R. Frühwirth, A. Strandlie, and T. Todorov. Reconstruction of Electrons with the Gaussian-Sum Filter in the CMS tracker at LHC. CMS-CR-2003-012.

[22] S. Agostinelli et al. GEANT4: A simulation toolkit. *Nuclear Instruments and Methods in Physics Research A*, 506:250–303, 2003.

[23] N. Arkani-Hamed, A. G. Cohen, E. Katz, and A. E. Nelson. *Phys. Lett. B*, 513:232, 2001.

[24] D. S. Bailey et al. FAMOS, CMS Fast Simulation. http://cmsdoc.cern.ch/famos/.

[25] M. Bengtsson and P. Roivainen. Using the Potts glass for solving the clustering problem. *International Journal of Neural Systems*, 6:119–132, 1995.

[26] D. R. Bickel and R. Frühwirth. On a fast estimation of the mode: comparisons to other robust estimators with applications. 2003. under review.

[27] P. Billoir and S. Qian. Fast vertex fitting with a local parametrization of tracks. *Nuclear Instruments and Methods in Physics Research A*, 311:139–150, 1992.

[28] M. Blatt, S. Wiseman, and E. Domany. Super-paramagnetic clustering of data. *Phys. Rev. Lett.*, 76:3251–3254, 1996.

[29] R. Brun et al. ROOT. http://root.cern.ch.

[30] M. Cavaglià, S. Das, and R. Maartens. Will we observe black holes at the LHC? *Class. Quantum Grav.*, 20 No 15, 2003.

[31] The Large Hadron Collider, conceptual design. Technical report.

[32] A. Chamblin and G. C. Nayak. Black Hole Production at LHC: String Balls and Black Holes from pp and Lead-lead Collisions. *Phys. Rev. Lett.*, D66, 2002.

[33] C. Charlot et al. CMS Simulation Facilities. Technical Report CMS TN 93-63. `http://cmsdoc.cern.ch/cmsim/cmsim.html`.

[34] CMS Collaboration. Addendum to CMS Tracker TDR. Technical Report CERN/LHCC 2000-016.

[35] CMS Collaboration. CMS — Technical Homepage. `http://cmsdoc.cern.ch/cms.html`.

[36] CMS Collaboration. The Compact Muon Solenoid. `http://cmsinfo.cern.ch`.

[37] CMS Collaboration. Object oriented Simulation for CMS Analysis and Reconstruction. `http://cmsdoc.cern.ch/oscar`.

[38] CMS Collaboration. ORCA, CMS OO Reconstruction. `http://cmsdoc.cern.ch/orca`.

[39] CMS Collaboration. The Compact Muon Solenoid. technical proposal. Technical Report CERN/LHCC 96/45, CERN 1996.

[40] CMS Collaboration. The tracker project, technical design report. Technical Report CERN/LHCC 98-6 CMS TDR 5, 15 April 1998.

[41] S. Cucciarelli, M. Konecki, D. Kotlinski, and T. Todorov. Track-Parameter Evaluation and Primary-Vertex Finding with the Pixel Detector. CMS-NOTE-2003-026.

[42] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. Roy. Statist. Soc. B*, B 39:1–38, 1977.

[43] J. D'Hondt, P. Vanlaer, W. Waltenberger, and R. Frühwirth. Sensitivity of robust vertex fitting algorithms. Technical Report CMS Note 2004/002, 2004.

[44] S. Dimopoulos and G. Landsberg. Black holes at the LHC. *Phys. Rev. Lett.*, 87:161602, 2001. arXiv:hep-ph/0106295.

[45] S. Dimopoulos and Roberto Emparan. String Balls at the LHC and Beyond. *Phys. Lett.*, B526:393–398, 2002.

[46] K. V. Driessen and P. Rousseuw. Computing LTS regression for large data sets. Technical report, University of Antwerp, 1999.

[47] N. Estre. *Caractérisation des détecteurs silicium, recherche de vertex et étude du potentiel de découverte d'un boson de Higgs chargé léger dans l'expérience CMS.* PhD thesis, Université Claude Bernard - Lyon 1, 2004.

[48] N. Estre, E. Chabanat, and P. Vanlaer. Deterministic Annealing for Vertex Finding at CMS. Technical Report CMS Note, in preparation.

[49] P. Filzmoser. private communication.

[50] R. Frühwirth. Application of Kalman filtering to track and vertex fitting. *Nuclear Instruments and Methods in Physics Research A*, 262:444, 1987.

[51] R. Frühwirth et al. Vertex reconstruction and track bundling at the lep collider using robust algorithms. *Computer Physics Comm.*, 96:189–208, 1996.

[52] R. Frühwirth, M. Regler, R. Bock, H. Grote, and D. Notz. *Data Analysis Techniques for High-Energy Physics.* Cambridge University Press, Cambridge, 2000.

[53] R. Frühwirth and T. Speer. A Gaussian-Sum Filter for Vertex Reconstruction. CMS-CR-2004-002.

[54] R. Frühwirth and A. Strandlie. Track fitting with ambiguities and noise: a study of elastic tracking and nonlinear filters. *Computer Physics Communications*, 120:197–214, 1999.

[55] R. Frühwirth, A. Strandlie, W. Waltenberger, and J. Wroldsen. A review of fast circle and helix fitting. *Nuclear Instruments and Methods in Physics Research A*, 502:705–707, 2003.

[56] R. Frühwirth and W. Waltenberger. Adaptive Multi-vertex fitting. Proc. 14th Int. Conf. on Computing in High Energy Physics (CHEP 04), Interlaken, Switzerland, 2004. HEPHY-PUB-798/04.

[57] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns — Elements of Reusable Object Oriented Software.* Addison-Wesley, 1995.

[58] J. E. Garcia. Little Higgs searches at LHC. 2004.

[59] GEANT3 Collaboration. GEANT3. `http://wwwasdoc.web.cern.ch/wwwasdoc/geant_html3/geantall.html`.

[60] GEANT4 Collaboration. GEANT4. `http://cern.ch/geant4`.

[61] A. Gersho and R. M. Gray. *Vector Quantization And Signal Compression.* Kluwer, Boston, 1992.

[62] F. Gianotti. Physics beyond the Standard Model at LHC. Talk given at the Physics at LHC conference, Vienna, july 2004.

[63] S. B. Giddings and S. Thomas. High energy colliders as black hole factories: The end of short distance physics. *Phys. Rev. D*, 65:056010, 2002. arXiv:hep-ph/0106219.

[64] D. Horn and A. Gottlieb. Algorithm for Data Clustering in Pattern Recognition Problems Based on Quantum Mechanics. *Phys. Rev. Lett.*, 88(1), 2002.

[65] R. M. Hristev. Matrix Techniques in Artificial Neural Networks. Master's thesis, University of Canterbury, Christchurch, New Zealand, 2000. alternate title: "Matrix ANN", available in electronic format.

[66] V. Innocente et al. COBRA, Coherent Object-oriented Base for Reconstruction, Analysis, and simulation. `http://cobra.web.cern.ch/cobra`.

[67] V. Innocente, L. Silvestris, and D. Stickland. CMS Software Architecture: Software Framework, services and persistency in high level trigger, reconstruction and analysis. *Computer Physics Communications*, page 140, 2001.

[68] R. Itoh and S. Ichizawa. Status of BELLE event processing framework based on a SMP-server cluster system. 1995. BELLE-NOTE-07.

[69] D. J. Jackson. A topological vertex reconstruction algorithm for hadronic jets. *Nuclear Instruments and Methods in Physics Research A*, 388:247–253, 1997.

[70] V. Karimäki. Effective vertex fitting. Technical Report CMS Note 1997/051.

[71] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69, 1982.

[72] S. Kundu. Gravitational clustering: a new approach based on the spatial distribution of the points. *Pattern Recognition*, 32:1149–1160, 1999.

[73] A. Leike. The phenomenology of extra neutral gauge bosons. *Phys. Rept.*, 317:143–250, 1999.

[74] M. Liendl. *Design and Implementation of an XML based object-oriented Detector Description Database for CMS.* PhD thesis, Vienna University of Technology, 2003.

[75] M. Lindström. Track reconstruction in the ATLAS detector using elastic arms. *Nuclear Instruments and Methods in Physics Research A*, 357:129–149, 1995.

[76] W. Mitaroff, G. Richter, and W. Waltenberger. Detector-Independent Vertex Reconstruction Toolkit VERTIGO. Proc. 14th Int. Conf. on Computing in High Energy Physics (CHEP 04), Interlaken, Switzerland, 2004. HEPHY-PUB-797/04.

[77] W. Mitaroff and W. Waltenberger. A Vertex Reconstruction Toolkit and Interface to Generic Objects (VERTIGO). Proc. 7th Int. Conf. on Linear Colliders (LCWS 04), Paris, 2004, LCnote LC-TOOL-2004-017.

[78] N. Polonsky. Supersymmetry — Structure and Phenomena. *Lect. Notes Phys.*, M68:1–169, 2001. Report-no: MIT-CTP-3164.

[79] K. Prokofiev. A kinematic fit and a decay chain reconstruction library. CMS-AN-2004-020.

[80] L. Randall and R. Sundrum. A large mass hierarchy from a small extra dimension. *Phys. Rev. Lett.*, 83:3370–3373, 1999.

[81] T. G. Rizzo. Black Hole Production at the LHC: Effects of Voloshin Suppression. *JHEP*, 0202, 2002. SLAC-PUB-9127.

[82] K. Rose. Deterministic annealing for clustering, compression, classification, regression and related optimization problems. *Proc. IEEE*, 86 (num 11):2210–2239, 1998.

[83] K. Rose, E. Gurewitz, and G. C. Fox. Statistical mechanics and phases transitions in clustering. *Phys. Rev. Lett.*, 65 (num 8):945–948, 1990.

[84] P. J. Rousseeuw and A. M. Leroy. *Robust Regression and Outlier Detection.* John Wiley & Sons, New York, 1987.

[85] D. W. Scott. *Multivariate Density Estimation: Theory, Practice and Visualization.* Wiley and Kegan Paul, 1992.

[86] G. Segneri and F. Palla. Lifetime based b-tagging with CMS. Technical Report CMS Note 2002/046.

[87] T. Sjöstrand. Pythia — the Lund Monte Carlo Event Generator. `http://www.thep.lu.se/ torbjorn/Pythia.html`.

[88] T. Sjöstrand, P. Edén, C. Friberg, L. Lönnblad, G. Miu, S. Mrenna, and E. Norrbin. *Computer Phys. Commun.*, 135, 2001. LU TP 00-30, hep-ph/0010017.

[89] T. Speer, K. Prokofiev, and R. Frühwirth. Vertex Fitting with the Kalman Filter Formalism in the ORCA Reconstruction Program. Technical Report CMS IN 2003/008.

[90] M. Spira and P. M. Zerwas. Electroweak symmetry breaking and Higgs physics. 1997.

[91] D. Stickland. ORCA: The Design, Implementation, and Deployment of a Functional Prototype OO Reconstruction Software for CMS. *Proc. Computing in High Energy and Nuclear Physics CHEP'00*, 2000.

[92] A. Strandlie. *Adaptive methods for the reconstruction of charged tracks in a noisy environment.* PhD thesis, University of Oslo, 2000.

[93] A. Strandlie and R. Frühwirth. Error analysis of the track fit on the Riemann sphere. *Nuclear Instruments and Methods in Physics Research A*, 480:734, 2002.

[94] A. Strandlie, J. Wroldsen, and R. Frühwirth. Treatment of multiple scattering with the generalized Riemann sphere track fit. *Nuclear Instruments and Methods in Physics Research A*, 488:332, 2002.

[95] A. Strandlie, J. Wroldsen, R. Frühwirth, and B. Lillekjendlie. Particle tracks fitted on the Riemann sphere. *Computer Physics Communications*, 131:95, 2000.

[96] L. Taylor et al. IGUANA, Interactive Graphics for User ANAlysis. `http://iguana.cern.ch`.

[97] The LEP Collaborations and the LEP Electroweak Working Group and the SLD Heavy Flavour and Electroweak Groups. A Combination of Preliminary Electroweak Measurements and Constraints in the Standard Model. Technical Report CERN-EP/2001-098 (hep-ex/0112021).

[98] A. C. Villanueva, J. A. H. Morata, and J. J. G. Cadenas. RecPack, a general reconstruction tool-kit. Conference Record of IEEE-NSS 2003, Portland (USA), October 2003.

[99] T. Virdee. Requirements from experiments in year 1. Prepared for 12th Chamonix LHC Performance Workshop, Chamonix, France, 3–8 March 2003.

[100] C. S. Wallace and D. L. Dowe. Minimum Message Length and Kolmogorov Complexity. *Computer Journal*, 42:270–283, 1999.

[101] W. Waltenberger et al. New developments in vertex reconstruction for CMS. *Nuclear Instruments and Methods in Physics Research A*, 502:699–701, 2003.

[102] A. Weingessel, E. Dimitriadou, and K. Hornik. A Voting Scheme for Cluster Algorithms. In G. Krell, B. Michaelis, D. Nauck, and R. Kruse, editors, *Neural Networks in Applications, Proceedings of the Fourth International Workshop NN'99*, pages 31–37, Otto-von-Guericke University of Magdeburg, Germany, 1999.

[103] C. Weiser. b-tagging, review talk, 2004. `http://agenda.cern.ch/askArchive.php?base=agenda&categ=a041643&id=a041643s5t5/transparencies`.

[104] M. Winkler. *A comparative study of track reconstruction methods in the context of CMS physics.* PhD thesis, Vienna University of Technology, 2002.

[105] M. Zickler. Vergleich der Liebesdarstellung in Hartmanns von Aue "Iwein" und Wolframs von Eschenbach "Parzival". Master's thesis, Universität Wien, 2004.

It is not for nothing that the scholar invented the PhD thesis as his principal contribution to literary form. The PhD thesis is the perfect image of his world. It is work done for the sake of doing work - perfectly conscientious, perfectly laborious, perfectly irresponsible.

Archibald Macleish, "The Irresponsibles"

Curriculum vitae

Dipl.-Ing. Wolfgang Waltenberger
Gumplowiczstraße 1/4/21
1220 Wien

Geboren am 10. Dezember 1974 in Gmunden(A).
Verheiratet (Manja Zickler), ein Kind (Fabian, 21. Juli 2000).

Ausbildung, Berufspraxis

1980 – 1985	**Volksschule** Ebensee (A)
1985 – 1993	**Gymnasium** Bundesgymnasium Gmunden (A)
Aug 1991 – Jun 1992	**R. J. Reynolds High School,** Exchange student, Winston-Salem NC (USA)
Okt 1993 – Mai 2001	**Studium** Techn. Physik, Technische Universität Wien Diplomarbeit (März 2001): "Towards 2d Quantum Gravity With Fermions"
1998 – 2000	**Studienassistenz** Zentraler Informatikdienst der TU Wien
1998	**Linux-Trainer** am Wifi Wien
1995 – 1996	**Tutor** im EDV Labor für Architektur und Raumplanung, TU Wien
Jun 2001 – Dez 2001	**Computerphysiker** am Institut für Hochenergiephysik, Österr. Akademie der Wissenschaften
Dez 2001 – Okt 2004	**Dissertation** "Development of Vertex Finding "and Vertex Fitting Algorithms for CMS"
Aug/Sep 2003	**Local organizer** der **Cern School of Computing**, Krems an der Donau (A)
seit Jan 2004	**Webmaster** der Seite `http://www.teilchen.at`