CDF/DOC/TRACKING/CDFR/6047
Version 1.0 , July 24, 2002

Vxprim in Run II

Hartmut Stadie, Wolfgang Wagner, Thomas Müller
Institut für Experimentelle Kernphysik, Universität Karlsruhe
Hans Wenzel
Fermilab

Abstract

During Run I Vxprim was used to find primary vertices with high resolution. We have ported the code to the Run II framework. In this note we briefly describe the algorithm and explain the parameters used in the Vxprim AC++ module. We give an example how to use the underlying vertex fitter directly and introduce a derived class that allows to seed the primary vertex finding with a physics object. We also investigate the performance concerning resolution and efficiency of the algorithm using beam data and Monte Carlo events.

1 Introduction

Many analyses like life time measurements and analyses needing a b-tag require a good knowledge of the primary vertex position for each event. In most cases in Run I the position of the beam line was used to estimate the primary vertex position in x and y if the z coordinate was known. This method proved to be sufficient for most applications in b-physics. Vxprim [1] was used to find the primary vertex with a better precision than the beam width for events with a high multiplicity (e.g. $t\bar{t}$). To achieve this goal Vxprim fitted the primary vertex using reconstructed tracks. In Run 1 one method to fit the beam line on a run by run basis [2] used the primary vertices found by Vxprim. In addition in Run II primary vertex reconstruction will be an useful tool to distinguish different interactions in a single bunch crossing.

We have ported the Vxprim algorithm and the underlying vertexing functions [3] to C++. An AC++ module that calls the Vxprim algorithm is run in production. The results of this module are used to determine the beam line positions [4]. The code for the module resides in the VertexMods package [5]. The VertexFitter class that houses the vertexing algorithms can be found in the VertexAlg package [6]. The Vxprim module stores the found primary vertices in a VertexColl object [7].

2 The Algorithm

We use the same algorithm as in Run I[1]. Subsection 2.2 describes the fitting function `vxgtpr`. As described in subsection 3.1 this function can be used outside the Vxprim module by using the VertexFitter class directly.

2.1 The Track Selection in the Vxprim Module

The Vxprim module reads in CdfTrack[8] objects and applies cuts on the tracks. The Vxprim module allows to cut on the transverse momentum of the tracks and on their impact parameter [9]. At least two stereo and two axial super layers with at least six hits each have to be assigned to a COT track to accept this track. Up to now no cuts are performed on the hit content of silicon tracks.

2.2 Finding the Primary Vertex

The main task of the algorithm is to find the tracks that originate from the primary vertex and remove all other tracks from the vertex fit. Using a wrong track in the fit results in getting a primary vertex position and covariance matrix that is not compatible with the right vertex position. To remove the tracks that do not originate from the primary vertex the fitting procedure is iterative. It starts with fitting a vertex with all tracks that passed the track selection cuts. Then it loops over all the tracks and subtracts one track a time from the fit and calculates the χ^2 of this track with respect to the fitted vertex. If the highest χ^2 value for any of the tracks exceeds a specified value(*maxtrackchi2*) this track is removed from the track sample. Then all remaining tracks are used to fit a vertex and this pruning procedure is repeated. If all tracks pass the χ^2 cut, the tracks go through the same procedure again doing a vertex fit with steering of the track parameters this time. This pruning of the track collection stops if a specified minimum number of tracks is left or all tracks pass the χ^2 cut. A last vertex fit is done with the remaining tracks to find the primary vertex position.

2.3 The Vertex Selection in the Vxprim Module

In the Vxprim module the pruning is allowed to go down to *minn* - 1 tracks. If at least *minn* tracks are used in the final fit, none of the tracks has a χ^2 with respect to the vertex that is greater than *maxtrackchi2*. In this case a Vertex object with the results of the fit is created and appended to the event record as part of a VertexColl. There is the possibility to rerun this algorithm using all the tracks that have been dropped in the pruning to search for additional primary vertices. All tracks with a z_0 parameter in a specified window around the z coordinate of the previously found vertex will be removed from the sample beforehand to reduce combinatorics.

3 How to use Vxprim

3.1 Using the VertexFitter Class directly

The algorithm as described in subsection 2.2 is coded as a function named `vxgtpr` in the VertexFitter class. This class belongs to the VertexAlg package [6]. The fitting routines can be used directly as described in [10] where the VertexFitter class is used to fit the decay vertex of J/Ψ candidates. The code in Figure 1 instantiates a VertexFitter object and passes a track view to the fitter. The return value of `vxgtpr` is the number of tracks used in the final fit. The last step is to access the fit results from the VertexFitter object.

```
VertexFitter fitter;
//get the default tracks
Handle<CdfTrackView> mytracks;
CdfTrackView::defTracks(mytracks,"PROD");
//pass a CdfTrackView to the fitter
fitter.newTracks(*mytracks);
//call the primary vertex finding algorithm
int nused = fitter.vxgtpr(4);
//print the vertex
std::cout << "vertex:" << fitter.vertex() << std::endl;
```

Figure 1: Code to call the primary vertex finding algorithm directly using the VertexFitter class.

3.2 Using the Vxprim Module

To use Vxprim one has to add the Vxprim module to its reconstruction/analysis job. The first set of parameters that can be set in the talk-to of the module specify the track selection cuts:

- *minpt*: only tracks with a transverse momentum greater than *minpt* GeV will be selected. The default value is 0.5 GeV.

- *maxd0*: tracks with an absolute value of the impact parameter(D) greater than *maxd0* will be removed. The default value is 3.0 cm.

- *cot*: set true to use tracks found in the Central Outer Tracker. The default value is true.

- *svx*: set true to use tracks found by one of the silicon tracking strategies. The default value is true.

If both *cot* and *svx* are set to true the best available tracks will be used. More technical, *CdfTrackView::defTracks* will be used to get a track set. If *cot* is set true and *svx* is set false the module will use the segment linked tracks (algorithm id = 14) from *CdfTrackView::allTracks*.

The *zcut* parameter is closely related to two other parameters:

- *zcut*: if the z position is already known only tracks that are less then *zcut* times the error on z away in z will be used.

- *combine*: set true to use FastZVertex information. The default is false.

- *QPV_z_resol*: defines the error that should be assumed for the FastZVertex z position in cm.

- *repeatsearch*: if this parameter is set to true, all tracks that are not close to any already found vertex will be used in the next pass. This allows to search for more than one primary vertex. The default is true.

- *keepall*: set true to bypass the check that at least *minn* tracks have been used in the final fit.

- *scattertracks*: set true to add multiple scattering and energy loss effects to COT stand-alone tracks from the last COT measurement to the beam pipe. The default is true.

- *COTErrorScale*: set the scale for the covariance matrix of COT stand-alone tracks. The default is 2.25 in analogy to the value used in outside-in silicon tracking.

There are two parameters that are used in the algorithm:

- *maxtrackchi2*: if the worst track has a χ^2 with respect to the vertex greater than *maxtrackchi2* the track will be dropped. The default value is 30.

- *minn*: at least *minn* tracks have to have been used in the final fit to accept the vertex. The default value is four.

Appendix A gives examples how to use the Vxprim Module only with COT or silicon tracks. These are the parameters used in production to create the "cot_vertices" and "svx_vertices" vertex collections.

3.3 Seeded Vxprim

Basically all physics analyses need to relate their physics objects to the vertex of the primary hard interaction. One possible strategy is to search for a primary vertex independent of any particular object and then check whether the objects of the analysis are compatible with originating from that vertex. Using vxprim we believe that this approach will best be served by running the vxprim module upstream of your analysis as described in section 3.2.

A second approach is based on utilizing the physics objects to seed the search for a primary vertex. Typical examples for these objects we have in mind are high-p_t leptons from the decay of heavy gauge bosons, jets or photons. To serve this second vertexing strategy we have developed the SeededVxPrimFitter as an interface to the VertexFitter class described in section 3.1. The idea is that a user can hand over one of the above mentioned objects to the SeededVxPrimFitter and the program uses this information to prepare a set of tracks which are forwarded to the actual fitter to calculate a vertex.

In the case of high-p_t leptons the seeding proceeds via the z_0 of the associated track. The SeededVxPrimFitter collects all tracks which have a z_0 that is within a window of the seed-z_0:

$$|z_{0,track} - z_{0,seed}| < \Delta z \tag{1}$$

For jets and photons the criterion is based on the angle θ between the jet axis and the track momentum $\vec{p}_{track}$.

$$\arccos\left(\frac{\vec{j}\cdot\vec{p}_{track}}{|\vec{j}|\cdot|\vec{p}_{track}|}\right) < \theta_{cone} \tag{2}$$

where $\vec{j}$ is a vector pointing along the jet axis. Alternatively, an η-ϕ-space criterion can be used:

$$\sqrt{(\eta_{jet}-\eta_{track})^2+(\phi_{jet}-\phi_{track})^2} < \Delta R_{cone} \tag{3}$$

The code for seeding with tracks is implemented, the code for the jets and photons is yet under development.

In the following we discuss the implementation of the SeededVxPrimFitter in more detail. The class publicly inherits from the VertexFitter class and therefore establishes an "is-a" relationship. This pattern maintains the entire functionality of

the basic fitter while providing extensions which are meant to ease the usage of the fitter class for the standard applications mentioned above.

The SeededVxPrimFitter is included in the VertexAlg package [6]. To use the program one has to include the header file `"VertexAlg/SeededVxPrimFitter.hh"` and link against the VertexAlg library. The dependency on VertexAlg has to be added to your makefile.

The steering parameters of the algorithm are initialized with the `initialize()` member function which takes the following arguments:

- minpt: minimum p_t of the tracks handed over to the vertex fitter. Default is 0.5 GeV.
- $D_{0,max}$: maximum impact parameter D_0 with respect to the origin. Default is 3 cm.
- Δz: window to select tracks according to the criterion given in (1).
- minNtrack: minimum number of tracks required to be attached to a vertex. Default is 4. Integer values greater or equal to 3 are allowed.
- $\chi^2_{track,max}$: threshold of the pruning algorithm. Maximum allowed χ^2 of a track with respect to a vertex. Default is 12.25.
- combine: flag to enable the usage of a seed vertex for a constraint fit. Default is false.
- trackSet: number to characterize the algorithms of the tracks which are used to prepare a track set for the fitter. Numbers: 1, 2 and 3 are allowed. 1 means the tracks in the *defTracks* set made by the silicon outside-in or silicon stand-alone algorithms are used. 2 means all tracks from *defTracks* are used. 3 means only COT stand-alone tracks are used.

In a typical application it is foreseen that the `initialize()` function is called only once at the beginning of the analysis job. In the event loop the fit is performed by calling the member function `fitPVtx()` giving the reference to the CdfTrack object of the seed, e.g. the electron, as an argument.

If the z_0 of the primary interaction is known by some other source, e.g. the ZVertex collection, `fitPVtx()` can be called with giving the z_0 as an argument.

The results of the fit can be accessed as described in section 3.1. Additionally, the SeededVxPrimFitter provides a pointer to the input track collection and the number of input tracks using the access function `getResults()`. Example code how to use the SeededVxPrimFitter in an analysis module is given in appendix C.

3.4 Reading the VertexColl from Processed Data

The Vxprim module is run as a standard component in production. Three different collections of primary vertices are produced. The different VertexColl objects in the event record can be distinguished by their description:

- *DEFAULT_VXPRIM_VERTICES*: this VertexColl contains primary vertices reconstructed using the default track view(*defTracks*).
- *cot_vertices*: this VertexColl contains primary vertices reconstructed using COT tracks.
- *svx_vertices*: this VertexColl contains primary vertices reconstructed using silicon tracks.

The code fragment in Figure 2 shows how to access the VertexColl with primary vertices reconstructed using COT tracks.

```
EventRecord::ConstIterator iter(event,
  StorableObject::SelectByClassName("VertexColl") &&
  StorableObject::SelectByDescription("cot_vertices"));
 if (iter.is_valid())
   {
     //std::cout << "coll found" << std::endl;
     ConstHandle<VertexColl> vertexset(iter);
     const VertexColl::CollType contents = vertexset->contents();
     for(VertexColl::const_iterator vertex = contents.begin();
         vertex != contents.end();++vertex)
       {
         Link<Vertex> vtxlnk = *vertex ;
         std::cout << "vertex x: " << vertex->x() << std::endl;
       }
   }
```

Figure 2: Code to access a primary vertex collection created by the Vxprim module.

4 Validation with the Run I Code

To see that the Run II code gives the same results as the Run I version both programs processed the same events. We used Run I events and stripped off all track banks except the SVXS banks. In the Run I program we took out the part that deals with multiple scattering. The Run II job contained a little module that creates a

CdfTrackColl out of the SVXS banks. So both programs got the same tracks with the same track parameters and covariance matrices. By applying the same track cuts and using the same parameters inside Vxprim both versions produced the same results with respect to the expected (floating point) precision. Part of the output of this test is shown in appendix B.

5 Vxprim using Beam Data from Run II

We have looked at the primary vertex positions in run 144574 using production version 4.3.2a_01. The figures 3 and 4 show the distribution of the primary vertex positions in x respectively y for COT tracks. Only vertices with a z coordinate between -1 cm

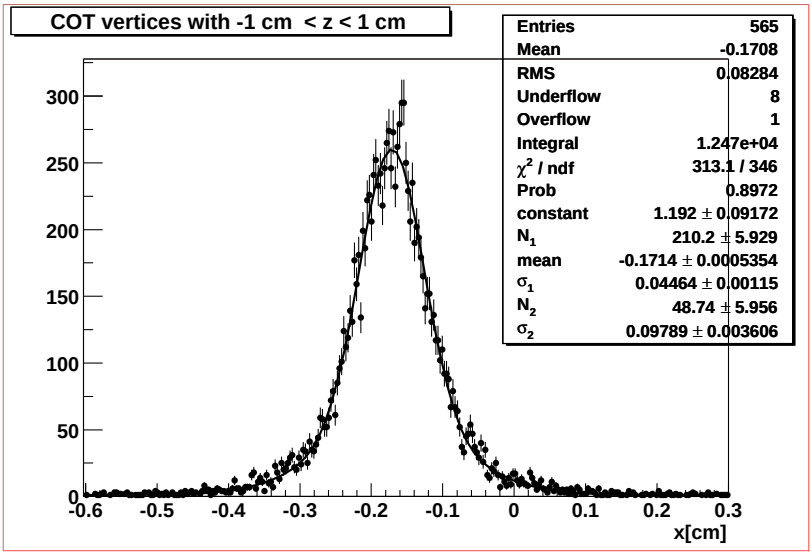

Figure 3: Distribution of the x-coordinate of primary vertices found by Vxprim using COT tracks.

and 1 cm have been used to reduce the effect of the slope of the beam line. One can see the beam profile convoluted with the Vxprim resolution. For COT tracks the distribution is clearly dominated by the Vxprim resolution (approximately 450 μm). The expected beam width is in the order of 30 μm [4]. The same plot for silicon tracks can be seen in the figures 5 and 6. Here the width of the peak is around 50 μm.

We have used run 138046 stream G data to make distributions for the number of tracks used in the final fit. Figure 7 shows the result for Vxprim using COT tracks. The mean number of tracks used in the fit is around 16 and far away from the cut value of four tracks. In Figure 8 for silicon tracks the situation is worse. There are many events that do not have enough tracks after the pruning to form a vertex. The mean number of tracks used in the fit is six. We explain this apparent discrepancy with the misalignment of the silicon detectors.

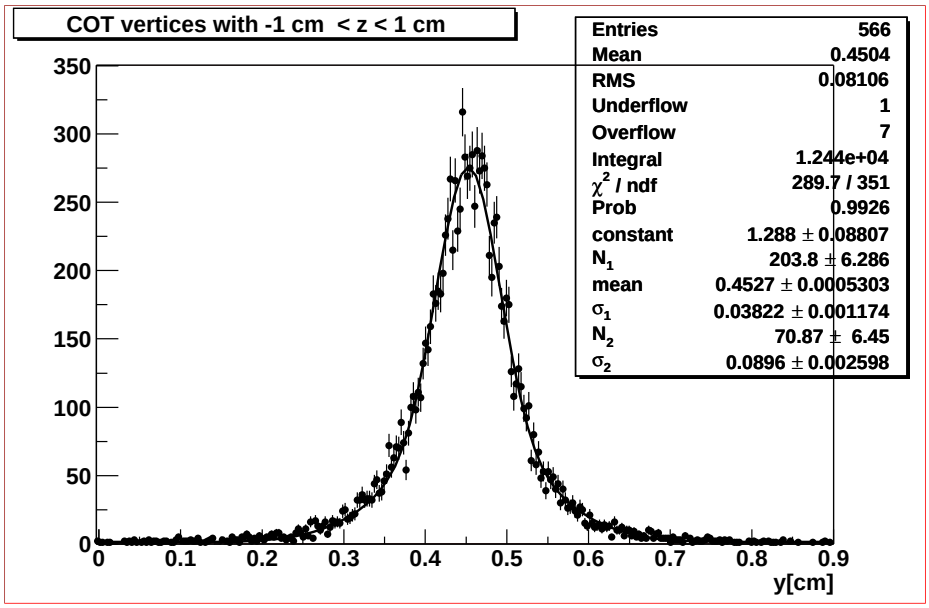

Figure 4: Distribution of the y-coordinate of primary vertices found by Vxprim using COT tracks.

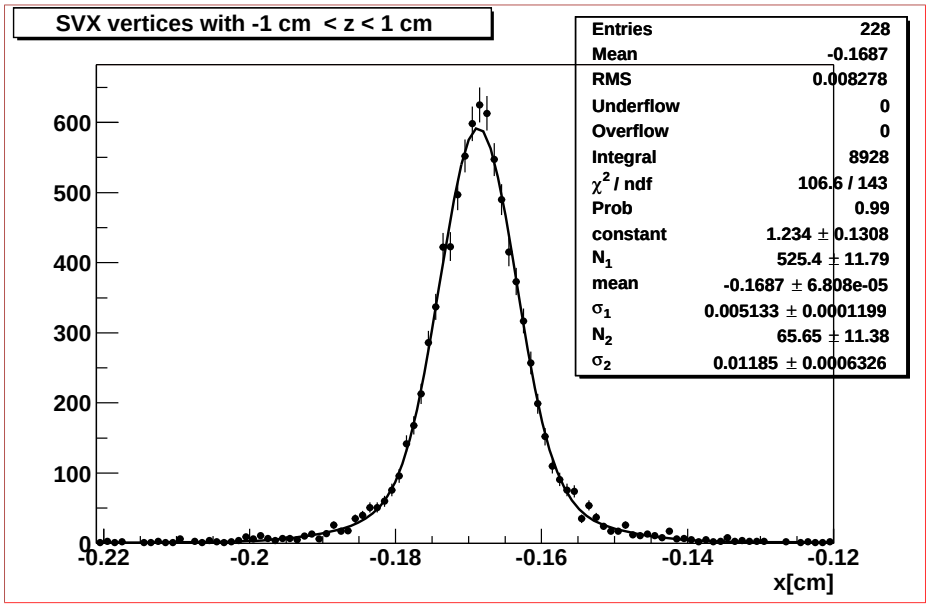

Figure 5: Distribution of the x-coordinate of primary vertices found by Vxprim using silicon tracks.

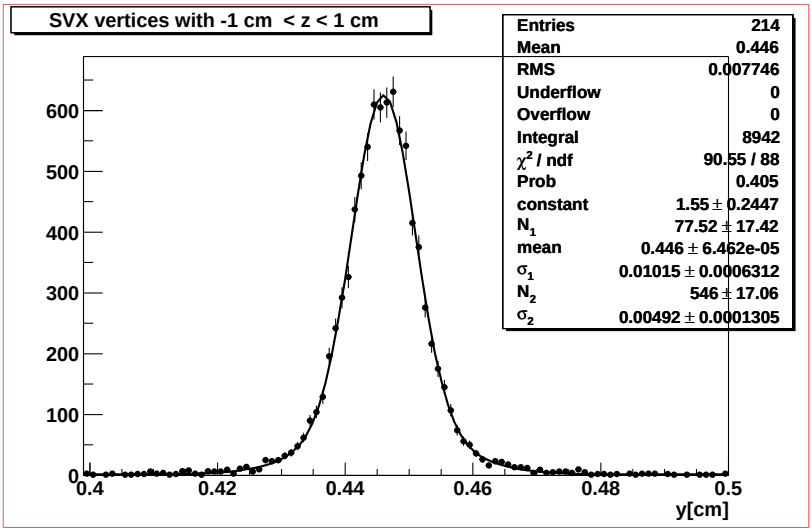

Figure 6: Distribution of the y-coordinate of primary vertices found by Vxprim using silicon tracks.

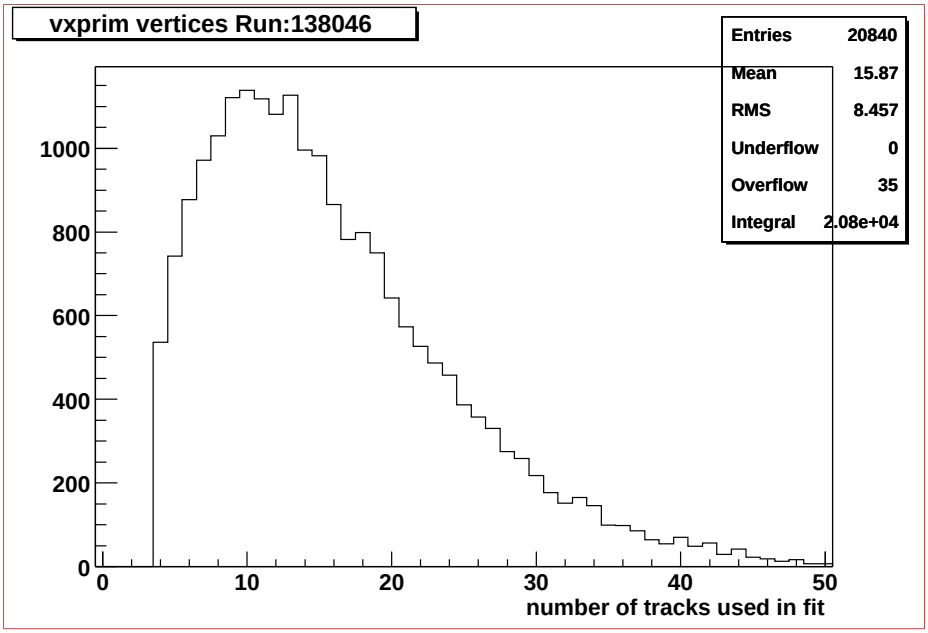

Figure 7: Distribution of the number of tracks used in the final vertex fit of the primary vertex using COT tracks.

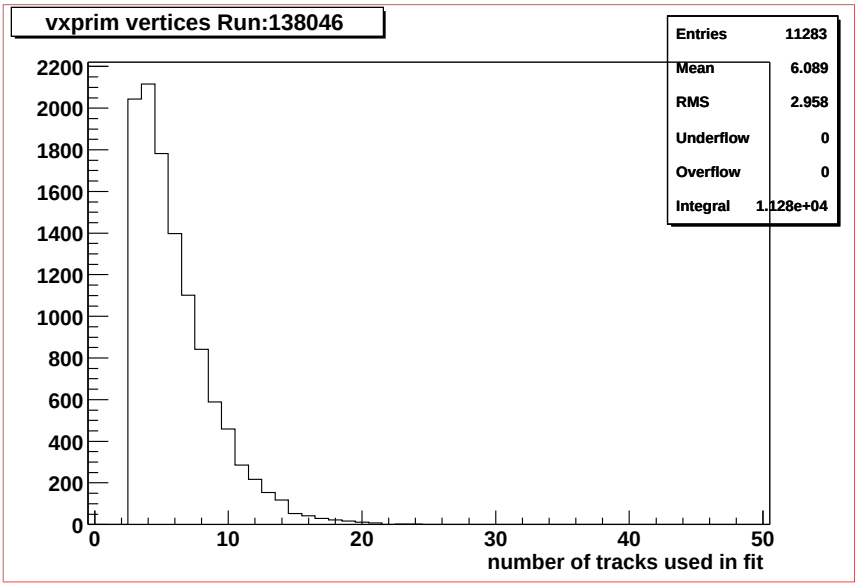

Figure 8: Distribution of the number of tracks used in the final vertex fit of the primary vertex using silicon tracks.

The goal of Vxprim is to find a primary vertex with a resolution below the beam width for events with a high track multiplicity and not to find a primary vertex for every event. Nevertheless we have investigated how often Vxprim finds a primary vertex. The percentage of events for which at least one vertex has been found in an event during production is shown in table 1 for different data streams and production versions. One sees good performance for COT tracks. The alignment and the silicon tracking are still being worked on, so we expect the numbers for silicon tracks to improve,as well as the numbers for Vxprim using the default tracks.

data file	cdfsoft version	default Vxprim	cot-Vxprim	svx-Vxprim
jr0234be.0031phys	4.3.2a_01	70.98%	94.69%	59.07%
jr0234be.0073phys	4.3.2a_01	70.90%	95.31%	59.62%
br0234be.002cphys	4.3.2a_01	73.31%	94.20%	58.17%
br0234be.009ephys	4.3.2a_01	73.31%	94.15%	57.22%
br0234be.002cphys	4.5.2_03	79.84%	94.49%	41.97%
br0234be.009ephys	4.5.2_03	80.24%	94.32%	41.79%

Table 1: The percentage of events for which at least one vertex has been found. The second column gives the name of the production version that has been used. The third column shows the results for the Vxprim module using the default track collection. The fourth column shows the results for the Vxprim module using COT tracks and the fifth shows the results for Vxprim using silicon tracks.

6 Tests with Monte Carlo Events

To study the Vxprim performance and the performance of the SeededVxprimFitter class in particular we have run the seeded fitter over 1000 $W + g \rightarrow e + \nu_e + g$ Monte Carlo (MC) events. The MC sample was generated using the version 6.203 of the PYTHIA event generator integrated in the CDF software framework. Event generation, detector simulation and reconstruction were done using the CDF software release 4.5.0int4.

In a first test, Vxprim was seeded with the z-Vertex taken from the Monte Carlo truth, which is provided in the CDF framework by the OBSV bank. A second test was performed by seeding the vertexing with the z_0 of the reconstructed electron track in the event.

The electron identification does not rely on Monte Carlo truth information. In the Monte Carlo sample no more than one electron per event can be identified. The electron identification proceeds as follows: The electromagnetic cluster (CdfEMObject) is required to have a minimum $E_t > 18.0$ GeV. We require the cluster to have a track attached. Further identification cuts are: Had/EM < 0.125 (ratio of hadronic energy over electromagnetic energy in the calorimeter), $E/p < 2.0$ and $isolation < 0.1$. Using those cuts we can successfully identify an electron in 521 events.

Before an attempt is made to fit a vertex an adequate track sample is produced by requiring certain quality criteria as described in subsection 3.3. As a basic sample we used all default tracks (*defTracks*) as output by the CDF tracking software. This track sample comprises COT stand-alone tracks, silicon outside-in tracks and silicon stand-alone tracks. As a z-window we used $\Delta z = 1.5$ cm. The tracks need to have $p_t > 0.3$ GeV. The maximum D_0 with respect to the origin is 3.0 cm. These track cuts result in a track selection efficiency of 93.3% in the first test (OBSV seeding) and 98.7% in the second test (electron seeding). In the second case the denominator of the efficiency is the number of events with an identified electron. We believe that the difference between the two track selection efficiencies is due to the electron identification yielding an event sample with a slightly harder track p_t spectrum and a higher track multiplicity.

For those events which pass the track selection criteria an attempt is made to fit a primary vertex using Vxprim. The pruning parameters of the algorithm are as follows: the minimum number of tracks is set to 3, the maximum χ^2 of a track with respect to the vertex is set to 12.25 ($3.5 \cdot 3.5$). If the pruning succeeds the algorithm stops when all tracks attached to the vertex have a χ^2 below this threshold. If the pruning fails, the algorithm runs down to the limit of the minimum number of required tracks. Out of these tracks there is still at least one which has a higher χ^2 value than the threshold. In practice, we let the algorithm run down to the minimum minus one and look at the number of tracks attached to the vertex. If this number is below the minimum, we know, the pruning failed. In our study those are events with two tracks attached to the vertex. Figure 9 shows the number of tracks attached to the vertex.

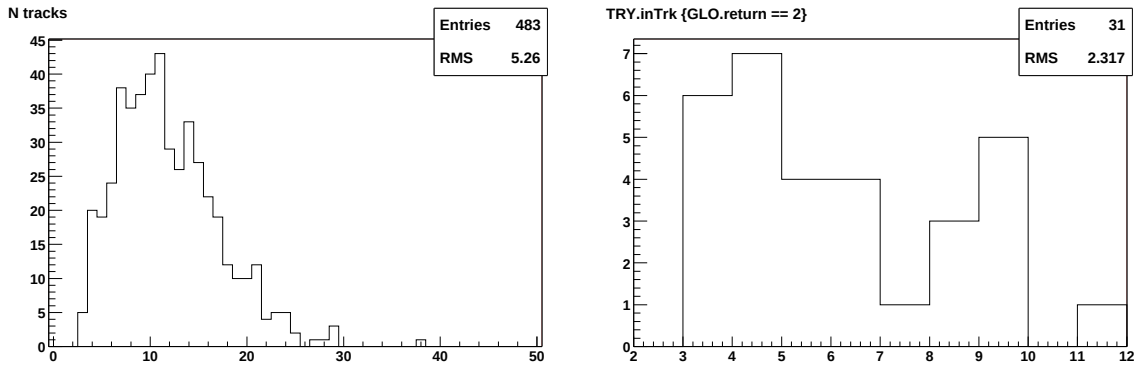

Figure 9: In the left plot we show the distribution of the number of tracks which are used to fit the found vertices.

On the left plot you see the events for which the pruning succeeded, on the right plot the events for which the pruning failed. It is expected to have some pruning failures in the 3,4 and 5 track bins. However, the shape of the distribution, in particular, the long tail to events with 7,8,9 and 10 input tracks is not understood and needs further investigation.

We define the vertexing efficiency as the ratio of events for which the vertex fit succeeded over those events which passed the track selection cuts. For our first test (OBSV seeding) we reach a vertexing efficiency of 93.0% and 94.0% for the second test.

An important issue of consideration is the vertex resolution we can expect with Vxprim. To get an estimate on what could be reached with a perfect detector we compare the reconstructed vertex positions with the MC truth. In Figure 10 the distributions of differences between reconstruction and MC truth are given. No major deviations between the two test cases are found. We find a typical resolution of 30 to 35 μm in the transverse coordinates x and y and a resolution of about 60 μm in z. The resolution is here quoted as the RMS value of the distribution.

We have also looked at the impact parameter resolution of the tracks attached to the vertex. The distribution of impact parameters is shown in the left plot of Figure 11. We find $\sigma_{imp} = 32$ μm (RMS). If assigning the tracks to different p_t bins we get the p_t-dependence of the impact parameter resolution. The result is shown in the right plot of Figure 11. The curve exhibits the expected $1/p_t$ dependence, starting with 50 μm for 0.5 GeV down to about 5 μm at $p_t > 10$ GeV. The horizontal error bars on pt are assigned by taking the RMS of the p_t for the six different bins. The vertical error bar is set to an estimated, fixed value of 2 μm. Fitting to the hypothesis $\sigma = a + b/p_t$ yields the coefficients $a = (5 \pm 2)$ μm and $b = (18 \pm 2)$ μm. Within the errors the expected functional form of the p_t dependence can be reproduced. The actual numbers for the parameters have to be taken cum grano salis. Realistic

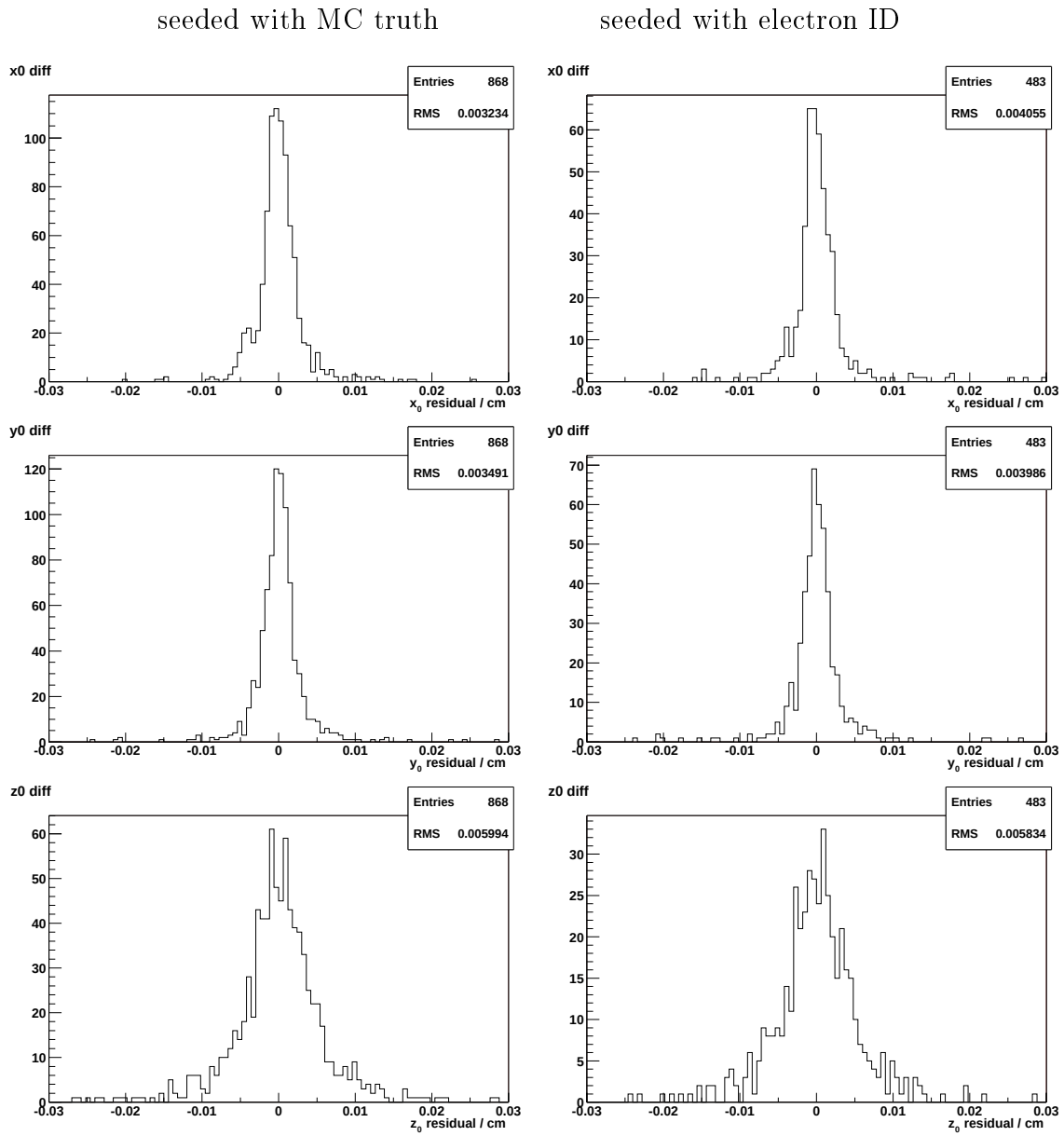

Figure 10: The plots show the difference Δx, Δy and Δz of the vertex coordinates taken from the Monte Carlo truth (as given by the OBSV bank) and the reconstructed vertex coordinates. The first column shows the results if the vertexing algorithm was seeded with the Monte Carlo truth. The second column shows the result if the vertex algorithm was seeded with the identified electron.

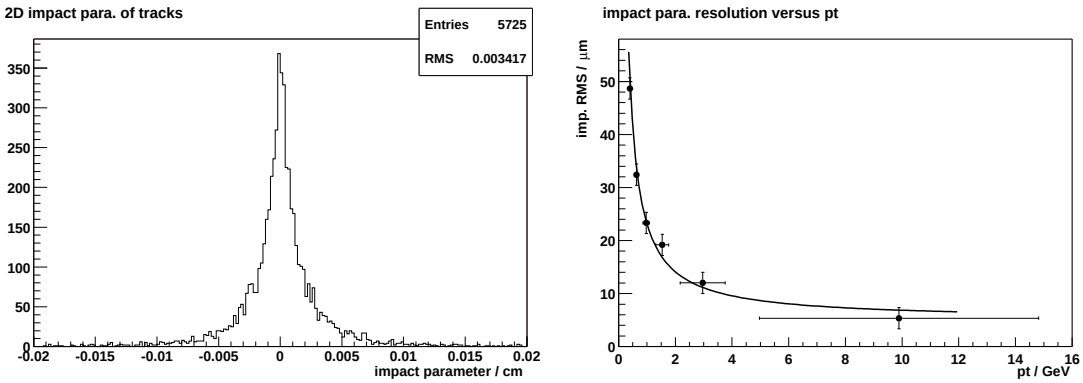

Figure 11: The left plot shows the impact parameter resolution of the tracks used for the vertex fit in simulated data. The vertex algorithm was seeded with the identified electron in the event. The right plot shows the average impact parameter resolution for 6 p_t bins in simulated data. First bin: $p_t < 0.5$ GeV; 2nd bin: $0.5\,\text{GeV} \leq p_t < 0.8$ GeV; 3rd bin: $0.8\,\text{GeV} \leq p_t < 1.2$ GeV; 4th bin: $1.2\,\text{GeV} \leq p_t < 2.0$ GeV; 5th bin: $2.0\,\text{GeV} \leq p_t < 5.0$ GeV; 6th bin: $p_t \geq 5.0$ GeV. The p_t value for each bin was obtained by averaging over the p_t of all input tracks for this bin.

numbers have to be extracted from data and not Monte Carlo events.

6.1 Impact Parameter Resolution in Collider Data

We run the analysis described above also on collider data and extracted the impact parameter resolution versus p_t. The result is shown in Fig. 12. The data are taken from the stripped high p_t inclusive electron sample, file `CentralEle_Inclusive_PR0713_0.dat`. The data were reconstructed with the CDF software version 4.5.2. The base sample was bhel03.

The fit yields: $a = (18 \pm 2)$ μm and $b = 30 \pm 3)$ μm, results quite different from the MC. The reason is probably that the amount of material is underestimated in the MC and that alignment is not simulated.

7 Conclusions

We have ported the Run I code of the primary vertex finding package Vxprim to the Run II framework. The algorithm can be used with an AC++ module as well as be called directly using the VertexFitter class. The ported code was validated against the Run I algorithm. The results are consistent within the expected floating point precision. Vxprim is used to calculate the beam line for CDF. The width of the central peak of the vertex distribution in x and y is on the order of 50 μm using only tracks with silicon information. This value includes the vertex resolution and the

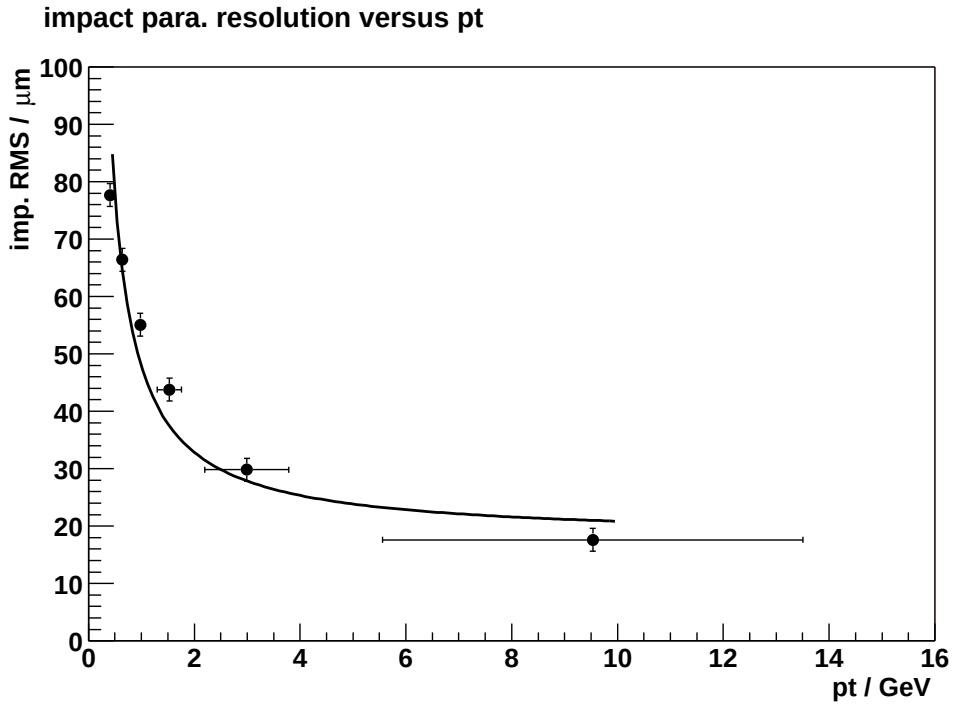

Figure 12: Average impact parameter resolution for 6 p_t bins in collider data.

beam width. We have tested Vxprim with MC $W \to e\nu$ events. The results suggest that a resolution of 40 μm (RMS) in x and y and 60 μm (RMS) in z can be achieved, assuming a perfectly aligned detector and no dead regions.

References

[1] F. Bedeschi *et. al.*, "A Primary Vertex Finding Package,". CDF Note-1789.

[2] H. Wenzel, "Fitting the beam position with the SVX,". CDF Note-1924.

[3] G. Punzi and S. Dell'Agnello, "The C$VTX Vertex-Finding Package,". CDF Note-1791.

[4] Run II Alignment group
http://www-cdf.fnal.gov/internal/upgrades/align/alignment.html.

[5] VertexMods source in the code browser
http://cdfcodebrowser.fnal.gov/CdfCode/source/VertexMods/.

[6] VertexAlg source in the code browser
http://cdfcodebrowser.fnal.gov/CdfCode/source/VertexAlg/.

[7] H. Stadie and H. Wenzel, "Proposal for a Run II Vertex Object,". CDF Note-5161.

[8] J. Boudreau and R. Snider, "A User's Guide to CdfTrack and Related Classes,". CDF Note-5089.

[9] H. Wenzel, "Tracking in the SVX,". CDF Note-1790.

[10] H. Stadie and H. Wenzel, "The inclusive B Lifetime Analysis in the Run II Software Framework,". CDF Note-5631.

A Example Talk-tos for Vxprim

Vxprim using COT tracks only:

```
module talk vxprim
  repeatsearch set f
  cot set t
  svx set f
  vertexcolldescription set cot_vertices
exit
```

Vxprim using silicon tracks:

```
module talk vxprim
  repeatsearch set f
  cot set f
  svx set t
  vertexcolldescription set svx_vertices
exit
```

B Output of the Validation with the Run I version

Run I Vxprim (no multiple scattering):

```
Run:       71022  Event:         472
tracks found:          42
tracks after cuts:           17
vertex:  -0.1189836     ,   9.8704264E-02 ,   -3.433281
ntracks:           13    CHI2:    41.20278
Run:       71022  Event:         529
tracks found:          51
tracks after cuts:           23
vertex:  -0.1207503     ,   0.1007950     ,   -4.527749
ntracks:           17    CHI2:    45.45284
```

Run II Vxprim on the same events(SVXS bank):

```
SVXSReadModule: 36 out of 42 added to EventRecord
Vxprim:
# of tracks after cuts:17
fit results: used tracks:13  chi2:41.1999
vertex:(-0.118984,0.098704,-3.43314)
SVXSReadModule: 45 out of 51 added to EventRecord
```

```
Vxprim:
# of tracks after cuts:23
fit results: used tracks:17  chi2:45.4389
vertex:(-0.120746,0.100795,-4.52783)
```

C Example Code to Use the SeededVxprimFitter Class

```
#include "VertexAlg/SeededVxprimFitter.hh"
...
SeededVxprimFitter _fitter;
...
MyAnalyis::beginJob()
{
  _fitter.initialize(3, 3.5*3.5, 1.0, false, 0.5, 3.0, false, 1);
  ...
}

MyAnalyis::event()
{
  CdfTrack& trk = findElectronFromWdecay();
  int usedTrk = _fitter.fitPVtx(trk);
  if (usedTrk > 0) {
    // Enough tracks found
    if (usedTrk >= 3) {
      // Pruning succeeded
      HepPoint3D vt = _fitter.vertex();
      HepSymMatrix cov = _fitter.vertexCovarianceMatrix();
    }
    else {
      // Pruning failed. However, the fit was performed with
      // the minimum number of tracks required.
      HepPoint3D vt = _fitter.vertex();
      HepSymMatrix cov = _fitter.vertexCovarianceMatrix();
    }
  }
  else {
    // Not enough tracks could be found to do the vertex fit.
  }
...
}
```