Hall A Software & Analysis Technical Details

Ole Hansen

Jefferson Lab

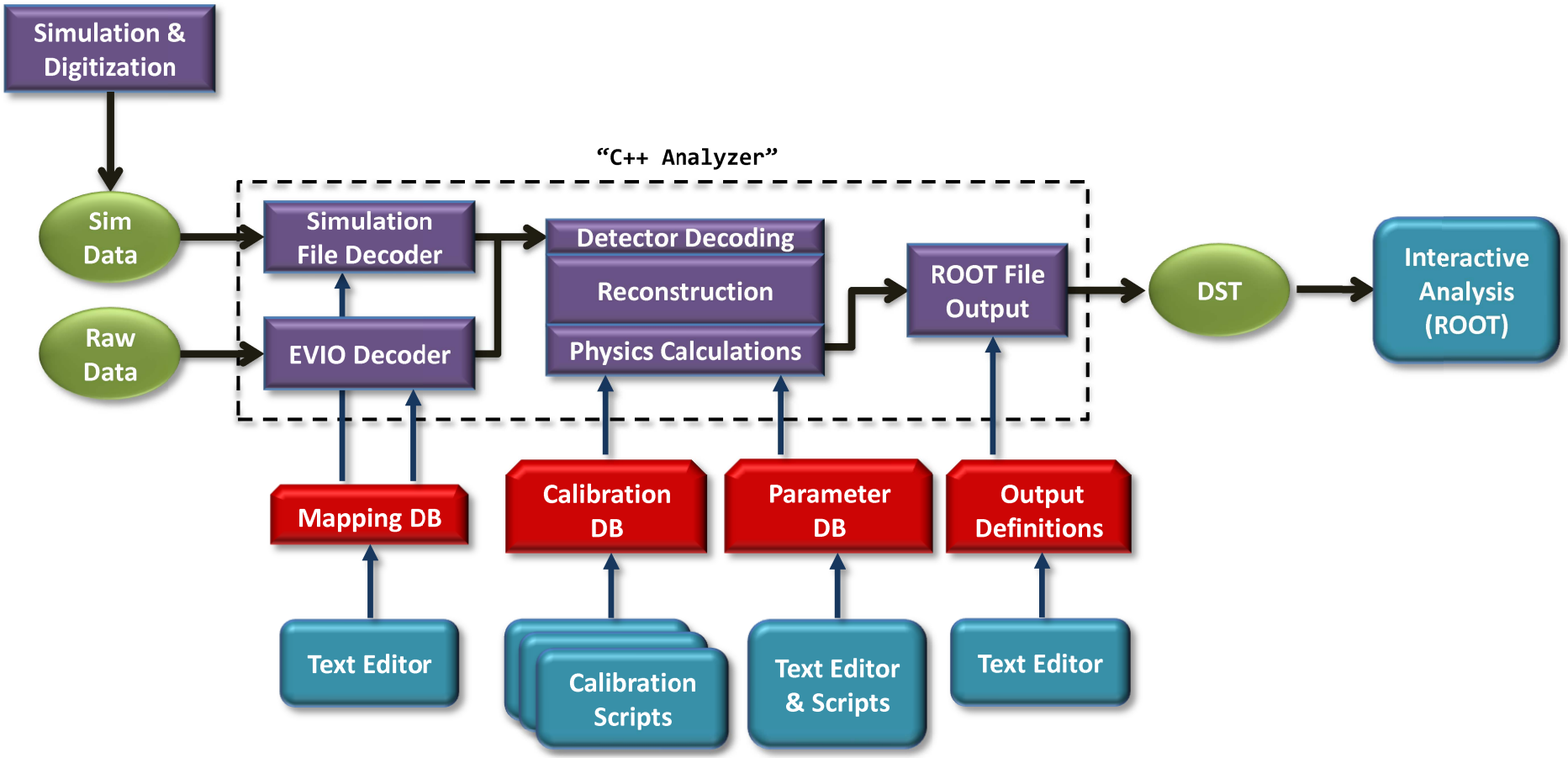
JLab 12 GeV Software Review
Afternoon Session
June 7, 2012

Jefferson Lab
Thomas Jefferson National Accelerator Facility

CEBAF@12 GeV

Contents

Hall A Data & Analysis Flow



C++ Analyzer Overview

- Standard Hall A analysis software since 2003
- Class library on top of ROOT
- Toolbox of analysis modules
- Predefined modules for generic analysis tasks and standard Hall A equipment
- Analysis controlled via interpreted or compiled C++ scripts
- Special emphasis on modularity
  - "Everything is a plug-in"
  - User code separate from core code
  - Load external user libraries dynamically at run time
  - Users write no more than the code really needed
  - Core analyzer suitable for fixed installation, like ROOT itself

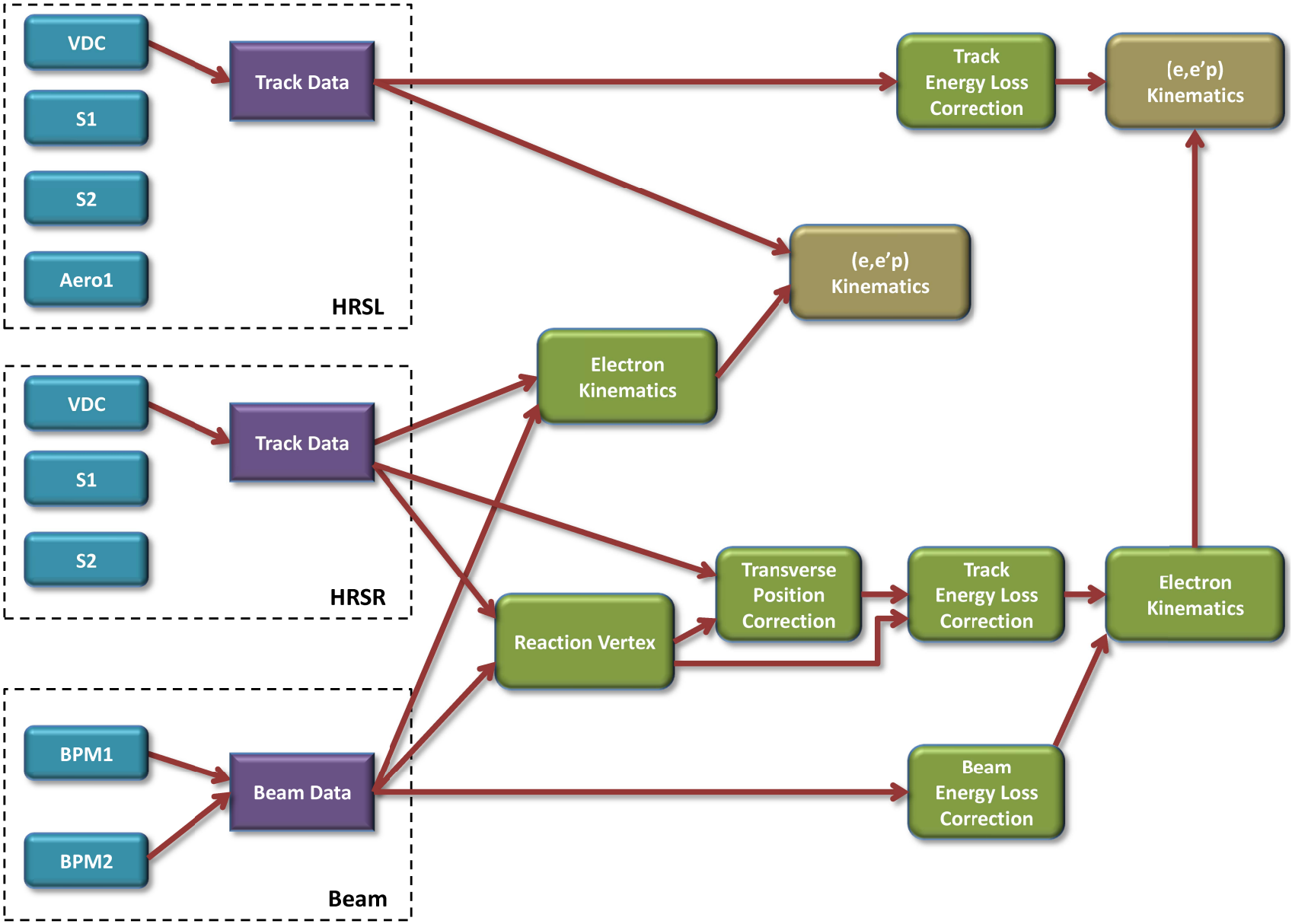
Analysis Objects

- Any class that produces "results"
- Every analysis object has unique name, *e.g.* R.s1
- Results stored in "global variables", prefixed with name, *e.g.* R.s1.nhits
- `THaAnalysisObject` common base class:
  - Support functions for database access
  - Support functions for global variable handling
- Actual objects implement various virtual functions
  - `DefineVariables()`
  - `ReadDatabase()`
  - `Init()`
  - etc.

Types of Analysis Objects

- "Detector"
  - Code/data for analyzing a type of detector.
    Examples: Scintillator, Cherenkov, VDC, BPM
  - Embedded in Apparatus or standalone
- "Apparatus" / "Spectrometer"
  - Collection of Detectors
  - Combines data from detectors
  - "Spectrometer": Apparatus with support for tracks and standard `Reconstruct()` function
- "Physics Module"
  - Combines data from several apparatuses
  - Typical applications: kinematics calculations, vertex finding, coincidence time extraction
  - Special applications: debugging, event display
  - Toolbox design: Modules can be chained, combined, used as needed

C++ Analyzer Example Physics Module Chain



C++ Analyzer User Interface

Example Replay Script

```
// Set up right arm HRS with the detectors we're interested in
THaHRS* HRSR = new THaHRS("R", "Right HRS");
//HRSR->AddDetector( new THaVDC("vdc", "Vertical Drift Chamber") ); // already in THaHRS
HRSR->AddDetector( new THaCherenkov("cer", "Gas Cherenkov counter" ));
HRSR->AddDetector( new THaShower("ps", "Pre-shower pion rej."));
HRSR->AddDetector( new THaShower("sh", "Shower pion rej."));
gHaApps->Add(HRSR);

// Ideal beam (perfect normal incidence and centering)
THaIdealBeam* ib = new THaIdealBeam("IB", "Ideal beam");
gHaApps->Add(ib);

// Simple kinematics and vertex calculations
Double_t mass_tg = 12*931.494e-3; // C12 target
THaPhysicsModule* EKR = new THaElectronKine("EKR","Electron kinematics R","R",mass_tg);
THaReactionPoint* rpr = new THaReactionPoint("rpr","Reaction vertex R","R","IB");
gHaPhysics->Add(EKR);
gHaPhysics->Add(rpr);

// The CODA data file we want to replay
THaRun* run = new THaRun("/rawdata/run_12345.dat");

// Set up and run standard analyzer (event loop)
THaAnalyzer* analyzer = new THaAnalyzer;
analyzer->SetOdefFile("HRSR.odef");   // Define output
analyzer->Process(run);               // Process all invents in the input
```

C++ Analyzer User Interface

Example Replay Script

```
// Set up right arm HRS with the detectors we're interested in
THaHRS* HRSR = new THaHRS("R", "Right HRS");
//HRSR->AddDetector( new THaVDC("vdc", "Vertical Drift Chamber") ); // already in THaHRS
HRSR->AddDetector( new THaCherenkov("cer", "Gas Cherenkov counter" ));
HRSR->AddDetector( new THaShower("ps", "Pre-shower pion rej."));
HRSR->AddDetector( new THaShower("sh", "Shower pion rej."));
gHaApps->Add(HRSR);

// Ideal beam (perfect normal incidence and centering)
THaIdealBeam* ib = new THaIdealBeam("IB", "Ideal beam");
gHaApps->Add(ib);

// Simple kinematics and vertex calculations
Double_t mass_tg = 12*931.494e-3; // C12 target
THaPhysicsModule* EKR = new THaElectronKine("EKR","Electron kinematics R","R",mass_tg);
THaReactionPoint* rpr = new THaReactionPoint("rpr","Reaction vertex R","R","IB");
gHaPhysics->Add(EKR);
gHaPhysics->Add(rpr);

// The CODA data file we want to replay
THaRun* run = new THaRun("/rawdata/run_12345.dat");

// Set up and run standard analyzer (event loop)
THaAnalyzer* analyzer = new THaAnalyzer;
analyzer->SetOdefFile("HRSR.odef");   // Define output
analyzer->Process(run);               // Process all invents in the input
```

C++ Analyzer User Interface

Example Replay Script

```
// Set up right arm HRS with the detectors we're interested in
THaHRS* HRSR = new THaHRS("R", "Right HRS");
//HRSR->AddDetector( new THaVDC("vdc", "Vertical Drift Chamber") ); // already in THaHRS
HRSR->AddDetector( new THaCherenkov("cer", "Gas Cherenkov counter" ));
HRSR->AddDetector( new THaShower("ps", "Pre-shower pion rej."));
HRSR->AddDetector( new THaShower("sh", "Shower pion rej."));
gHaApps->Add(HRSR);

// Ideal beam (perfect normal incidence and centering)
THaIdealBeam* ib = new THaIdealBeam("IB", "Ideal beam");
gHaApps->Add(ib);

// Simple kinematics and vertex calculations
Double_t mass_tg = 12*931.494e-3; // C12 target
THaPhysicsModule* EKR = new THaElectronKine("EKR","Electron kinematics R","R",mass_tg);
THaReactionPoint* rpr = new THaReactionPoint("rpr","Reaction vertex R","R","IB");
gHaPhysics->Add(EKR);
gHaPhysics->Add(rpr);

// The CODA data file we want to replay
THaRun* run = new THaRun("/rawdata/run_12345.dat");

// Set up and run standard analyzer (event loop)
THaAnalyzer* analyzer = new THaAnalyzer;
analyzer->SetOdefFile("HRSR.odef");   // Define output
analyzer->Process(run);               // Process all invents in the input
```

C++ Analyzer User Interface

Example Replay Script

```
// Set up right arm HRS with the detectors we're interested in
THaHRS* HRSR = new THaHRS("R", "Right HRS");
//HRSR->AddDetector( new THaVDC("vdc", "Vertical Drift Chamber") ); // already in THaHRS
HRSR->AddDetector( new THaCherenkov("cer", "Gas Cherenkov counter" ));
HRSR->AddDetector( new THaShower("ps", "Pre-shower pion rej."));
HRSR->AddDetector( new THaShower("sh", "Shower pion rej."));
gHaApps->Add(HRSR);

// Ideal beam (perfect normal incidence and centering)
THaIdealBeam* ib = new THaIdealBeam("IB", "Ideal beam");
gHaApps->Add(ib);

// Simple kinematics and vertex calculations
Double_t mass_tg = 12*931.494e-3; // C12 target
THaPhysicsModule* EKR = new THaElectronKine("EKR","Electron kinematics R","R",mass_tg);
THaReactionPoint* rpr = new THaReactionPoint("rpr","Reaction vertex R","R","IB");
gHaPhysics->Add(EKR);
gHaPhysics->Add(rpr);

// The CODA data file we want to replay
THaRun* run = new THaRun("/rawdata/run_12345.dat");

// Set up and run standard analyzer (event loop)
THaAnalyzer* analyzer = new THaAnalyzer;
analyzer->SetOdefFile("HRSR.odef");   // Define output
analyzer->Process(run);               // Process all invents in the input
```

C++ Analyzer User Interface

Example Replay Script

```
// Set up right arm HRS with the detectors we're interested in
THaHRS* HRSR = new THaHRS("R", "Right HRS");
//HRSR->AddDetector( new THaVDC("vdc", "Vertical Drift Chamber") ); // already in THaHRS
HRSR->AddDetector( new THaCherenkov("cer", "Gas Cherenkov counter" ));
HRSR->AddDetector( new THaShower("ps", "Pre-shower pion rej."));
HRSR->AddDetector( new THaShower("sh", "Shower pion rej."));
gHaApps->Add(HRSR);

// Ideal beam (perfect normal incidence and centering)
THaIdealBeam* ib = new THaIdealBeam("IB", "Ideal beam");
gHaApps->Add(ib);

// Simple kinematics and vertex calculations
Double_t mass_tg = 12*931.494e-3; // C12 target
THaPhysicsModule* EKR = new THaElectronKine("EKR","Electron kinematics R","R",mass_tg);
THaReactionPoint* rpr = new THaReactionPoint("rpr","Reaction vertex R","R","IB");
gHaPhysics->Add(EKR);
gHaPhysics->Add(rpr);

// The CODA data file we want to replay
THaRun* run = new THaRun("/rawdata/run_12345.dat");

// Set up and run standard analyzer (event loop)
THaAnalyzer* analyzer = new THaAnalyzer;
analyzer->SetOdefFile("HRSR.odef");   // Define output
analyzer->Process(run);               // Process all invents in the input
```

C++ Analyzer Output (DST)

- ROOT format
- Contents can be dynamically defined for each replay via input file

Example Output Definition File

```
# ----  Example e12345.odef  -------------

# Variables to appear in the tree.
variable L.s1.lt[4]
variable L.s1.rt

# The 'block' variables:  All data in Right HRS go to tree.
block R.*

# Formulas can be scalers or vectors.
# Lt4a is a scaler.
formula Lt4a 5.*L.s1.lt[4]

# Cuts can be defined globally and used in histograms.
# Cut C1 is a scaler.  Data is 0 or 1.
cut C1 L.s1.lt[4]>1350

# Histograms can involve formulas, variables, and cuts.
# TH1F, TH1D, TH2F, TH2D supported.
TH1F  rv1n  'L-arm vdc hits on V1'  L.vdc.v1.nhit  10 0 10
```

C++ Analyzer Databases

- Currently only flat text files supported
- Key/value pairs with support for scalars, arrays, matrices, strings
- Support for time-dependent values (essential!)
- History functionality available if files kept under version control (CVS)
- Plan to investigate relational database system, *e.g.* Hall D's

Example Database File

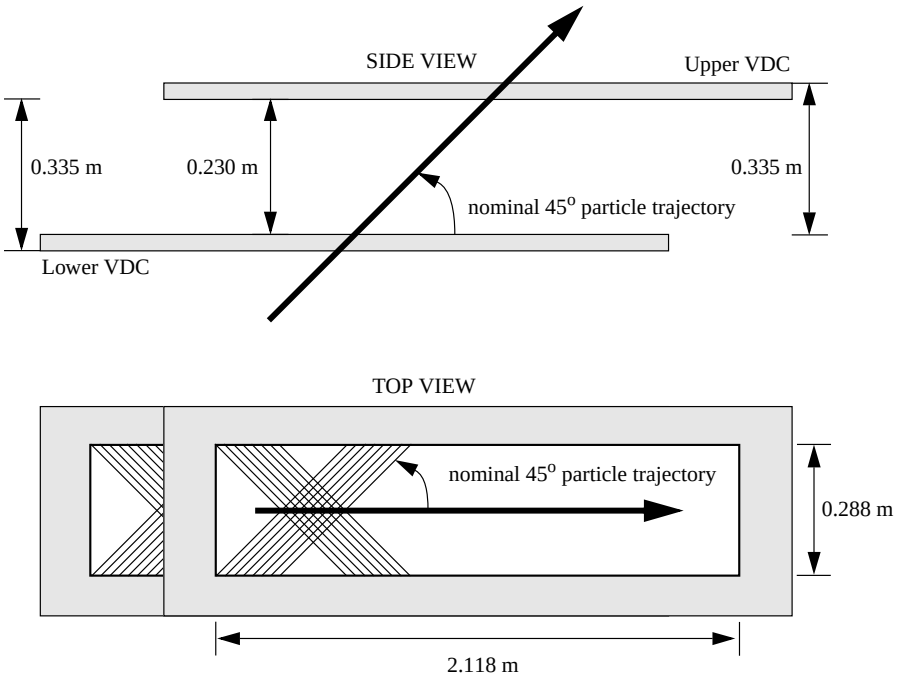
```
B.mwdc.planeconfig = u1 u1p x1 x1p v1 v1p \
                     u2 x2 v2 \
                     u3 u3p x3 x3p v3 v3p

# "Crate map":  crate slot_lo slot_hi model# resol   nchan
B.mwdc.cratemap =  3     6       21      1877   500     96  \
                   4     4       11      1877   500     96  \
                   4     17      24      1877   500     96

--[ 2008-02-31 23:59:45 ]
B.mwdc.maxslope      = 2.5

B.mwdc.size          = 2.0  0.5   0.0
B.mwdc.x1.size       = 1.4  0.35  0.0
```

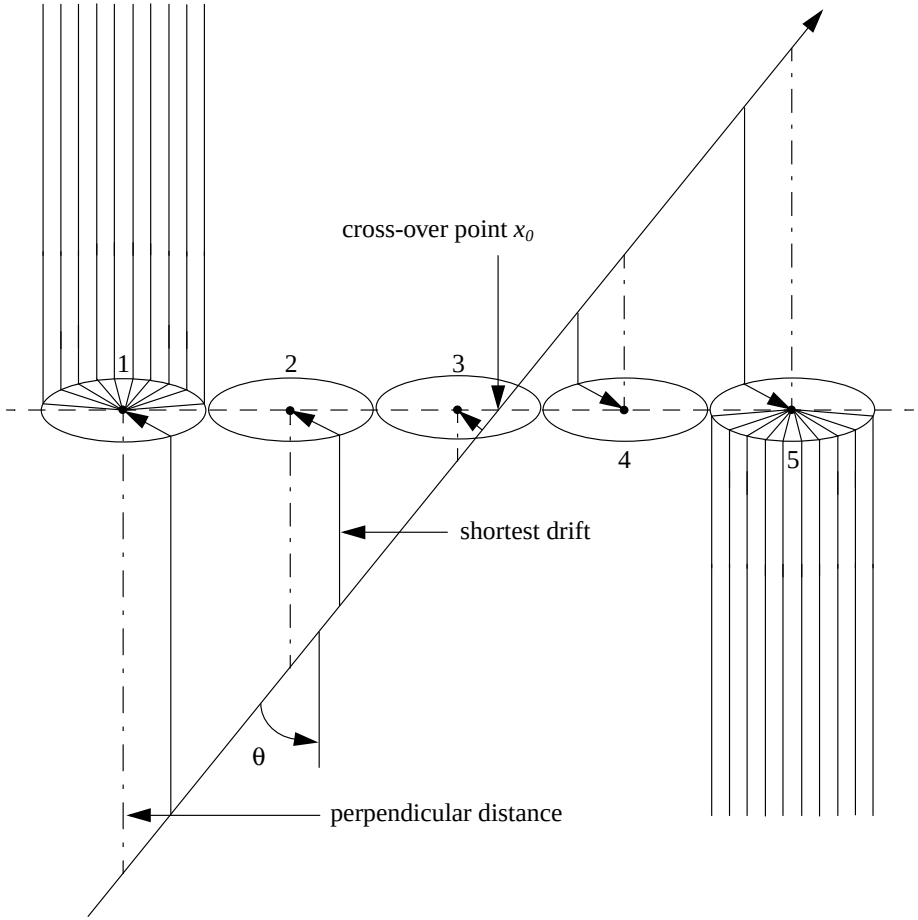
C++ Analyzer Limitations

- Essentially single-threaded
  - Plan to implement automatic event distribution to multiple "worker" subprocesses
  - Memory footprint not a major issue, but partially non-reentrant code is → `fork()` subprocesses instead of creating threads
- Support for 12 GeV DAQ environment not yet available
  - CODA 3 EVIO-library
  - Decoders for JLab 12 GeV pipelined electronics
  - *NB*: Event "re-assembler" for pipelined data streams to be provided by DAQ group. This is an essential component for running in pipelined mode (SBS and beyond).
- ROOT file output can be a performance bottleneck
  - Minor issue since ratio of reconstruction to output time is expected to rise with more demanding experiments
  - Mostly due to overbroad output definition (too many variables)
  - Coding bottlenecks probably correctable (known inefficiencies)

HRS VDC Tracking I

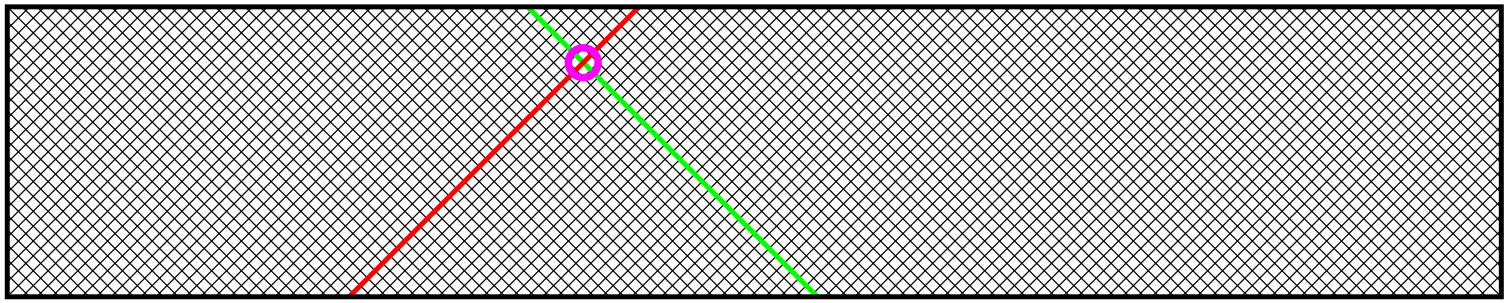
- *Vertical* Drift Chambers, optimized for precision measurement of single tracks
- Standard tracking systems for both HRSs
- Two wire directions (u and v), 368 wires per plane, 4.24 mm wire spacing





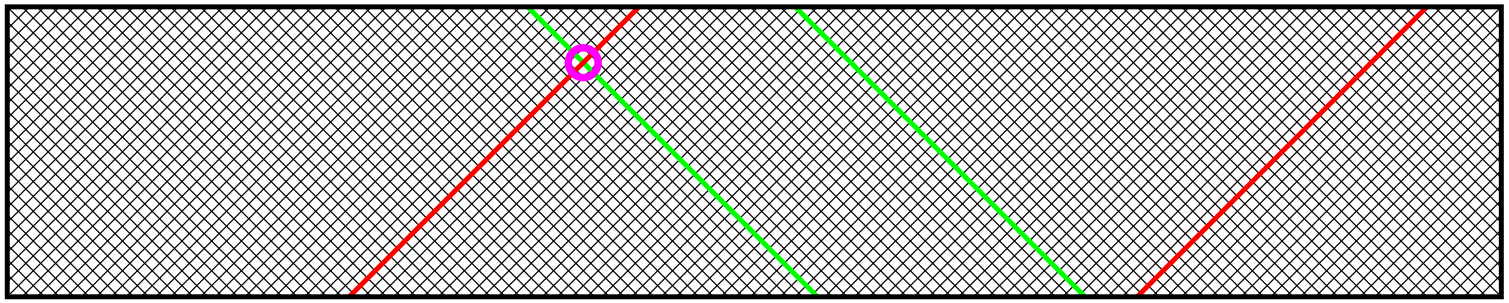
- Fit to avg. 5 independent time measurements per plane yield a position resolution of $\approx$ 225 μm FWHM

HRS VDC Tracking II: Algorithm



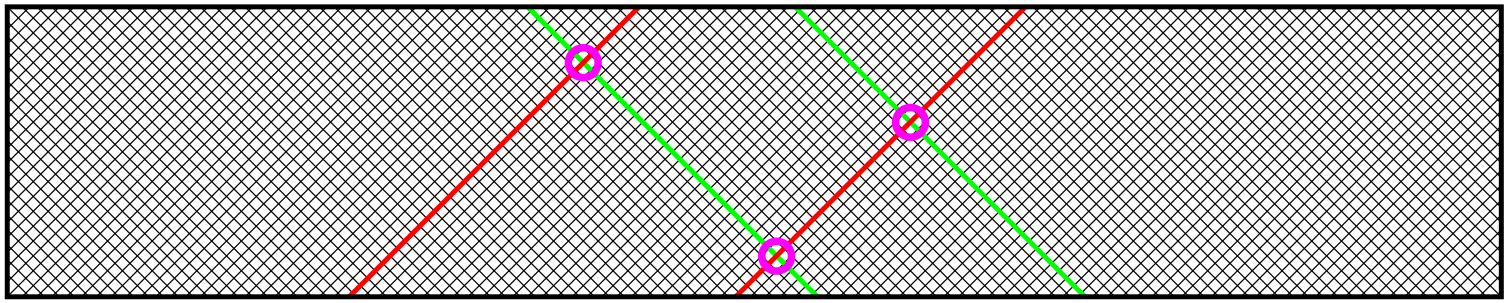
- Match u and v clusters in each chamber
  - Obvious if only one cluster per plane

HRS VDC Tracking II: Algorithm



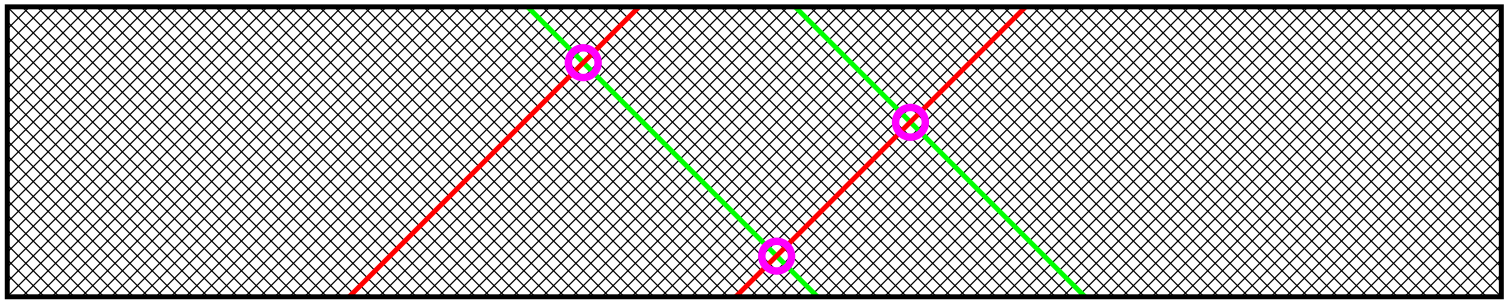
- Match u and v clusters in each chamber
  - Obvious if only one cluster per plane
  - If multiple clusters, geometrical/angle information may help

HRS VDC Tracking II: Algorithm



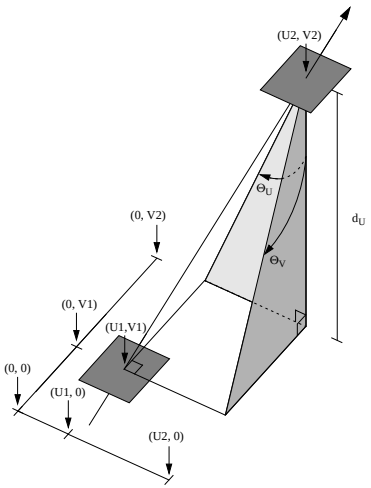
- Match u and v clusters in each chamber
  - Obvious if only one cluster per plane
  - If multiple clusters, geometrical/angle information may help
  - Advanced algorithm takes advantage of timing information form 3-parameter cluster fits to resolve most remaining ambiguities

HRS VDC Tracking II: Algorithm

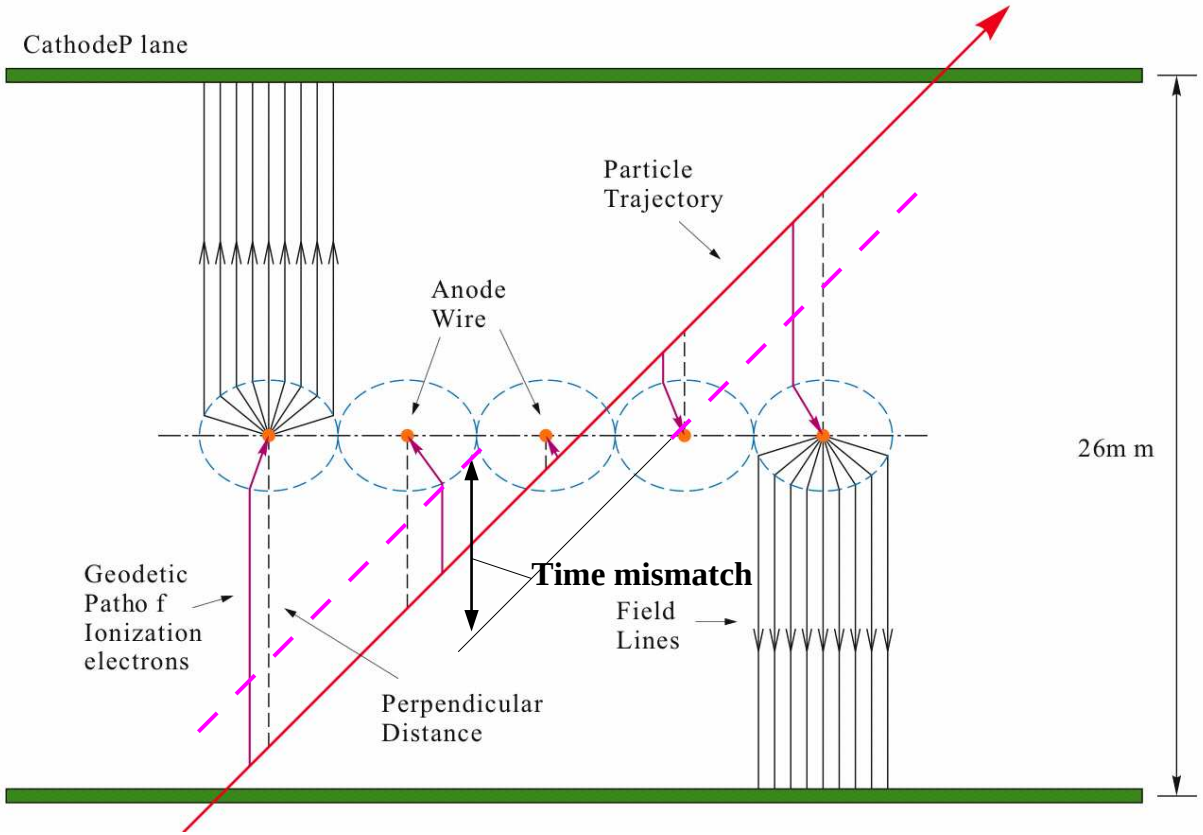


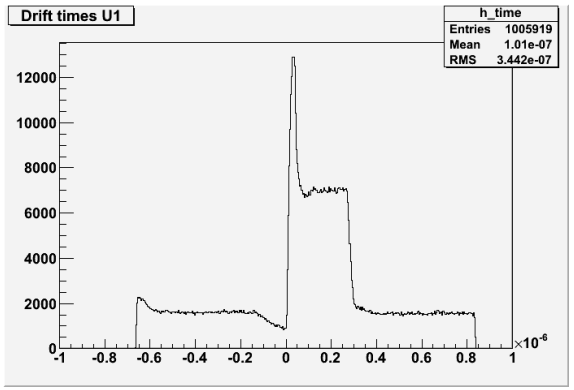
- Match u and v clusters in each chamber
  - Obvious if only one cluster per plane
  - If multiple clusters, geometrical/angle information may help
  - Advanced algorithm takes advantage of timing information form 3-parameter cluster fits to resolve most remaining ambiguities

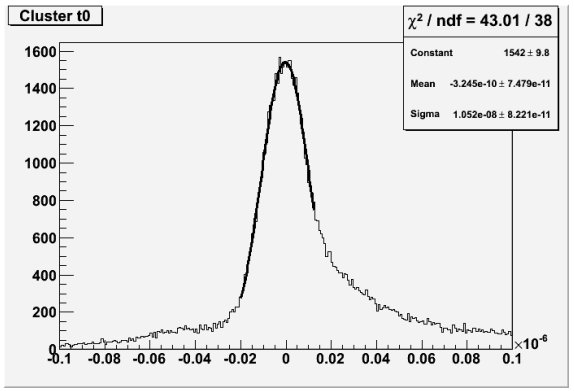
- Connect matched points in top and bottom, requiring consistent track angles
- Calculate track parameters at target by multiplying focal plane tracks variables with reverse transport matrix



HRS VDC Tracking III: 3-Parameter Cluster Fit



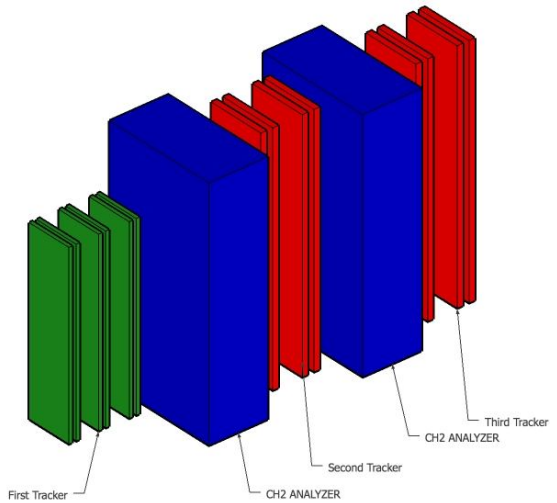


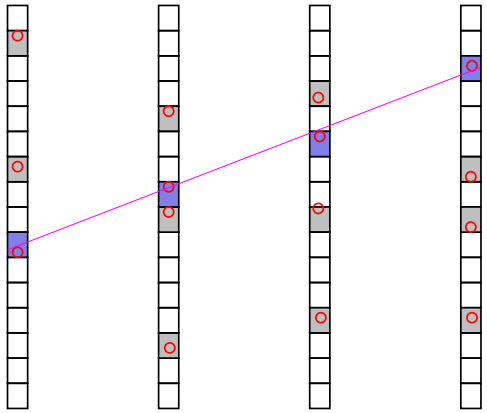


- Non-linear 3-parameter fit to extract track time offset t_0
- Computationally expensive: ca. $\times 20$ slower than 2-parameter fit
- $\approx$ 20 ns FWHM time resolution $\rightarrow$ background rejection factor $\approx$ 10-20
- Required for APEX: expect $\approx$ 2 accidental tracks per trigger
- Code written, still needs testing/debugging and integration

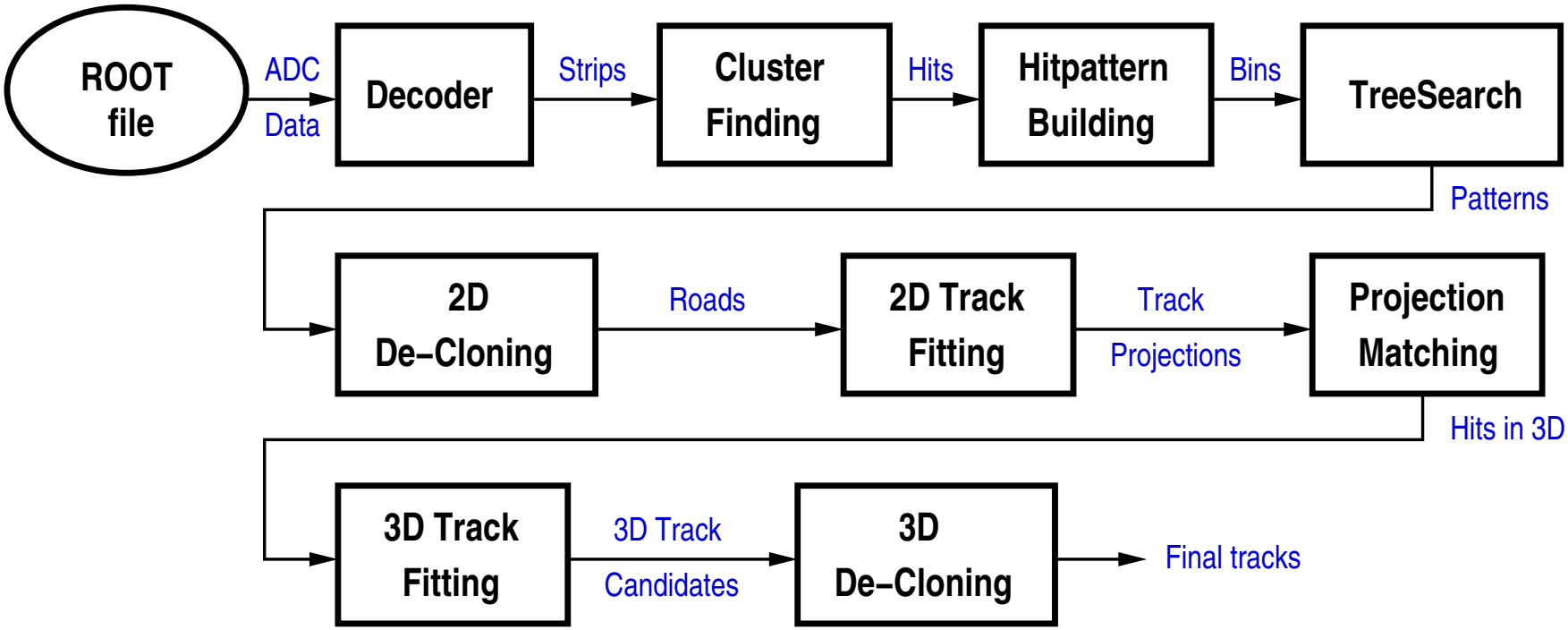
SBS Track Reconstruction Algorithm

- Challenge: GEM front trackers operating at up to 500 kHz/cm^2 count rate
- Reconstruction algorithm implemented in 2010/11 based on Hall A BigBite MWDC code
- APV25 decoder & analysis
  - Cluster finding
  - Peak fitting / noise rejection
- TreeSearch in coordinate projections $\longrightarrow$ roads
  - Very fast recursive template matching algorithm
  - Efficiently finds straight lines of hits (within configurable bin width)
  - Also used at HERMES and JLab (Qweak experiment)
- Correlation of roads from different projections, either
  - geometrically (3+ projections); or
  - via hit amplitude & timing in shared readout planes (2 projections)
  - Monte Carlo indicates that 2 projections are sufficient
- 3D fit of correlated hits



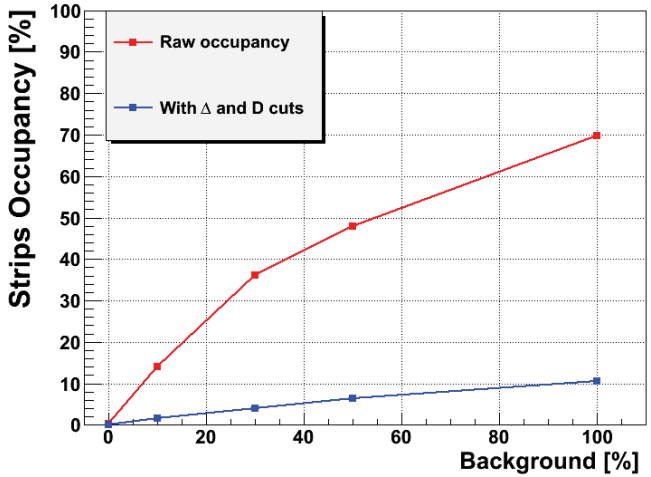


SBS Track Reconstruction Flow

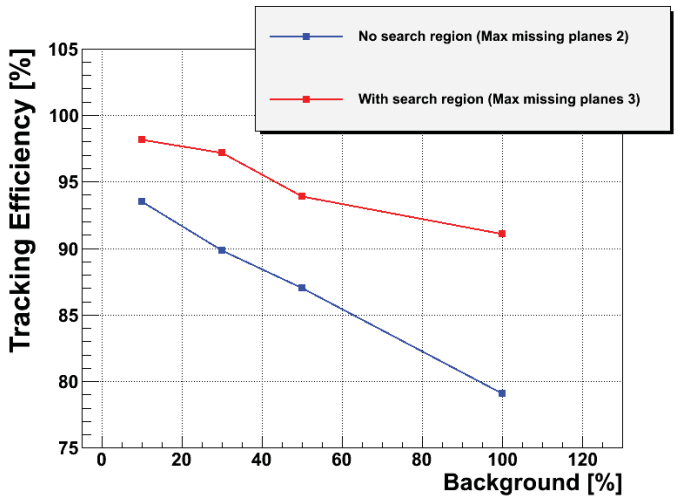


SBS Tracking Monte Carlo (with V. Mamyan, CMU)

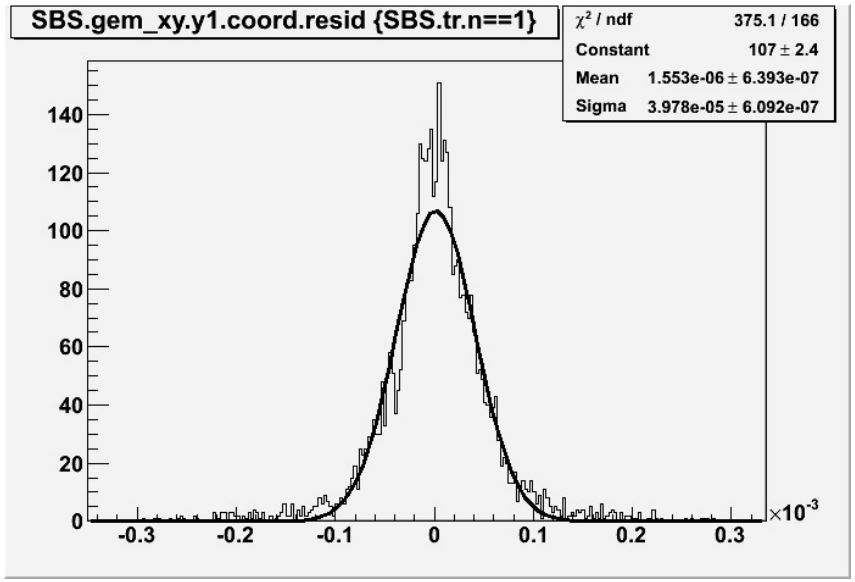
Front tracker GEM strip occupancy



Tracking Efficiency



Track reconstruction accuracy



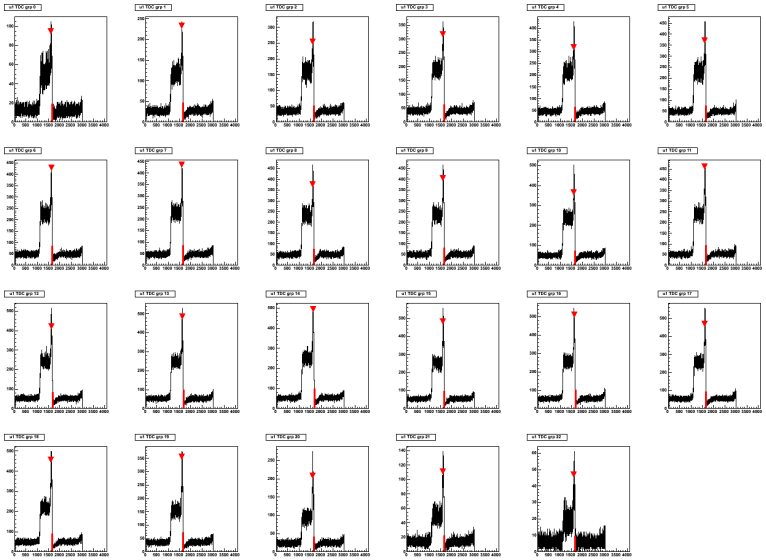
- Realistic digitization of GEM & electronics response
- $>$ 90% tracking efficiency despite $>$ 70% raw occupancy!
- $\approx$ 40 μm reconstruction accuracy
- Needs further testing, esp. with real data

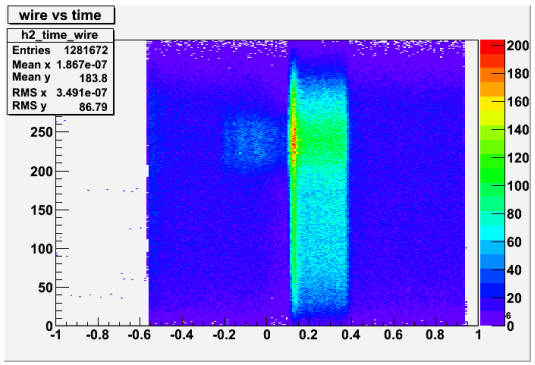
HRS Optics Calibrations

- Requirements
  - Optics runs with sieve slit collimator and multifoil target
  - Multi-dimensional minimization procedure for reconstructed residuals
- General-purpose minimization software package available
  - `optimize++`
  - ROOT application, written in C++
  - Uses Minuit as underlying minimization engine
  - Very well tested over almost a decade for a broad variety of experimental configurations (optics tunes)
- Optics calibration of BigBite and other spectrometers not as well tested and no general tool readily available, but similar principle

VDC Calibration Tools

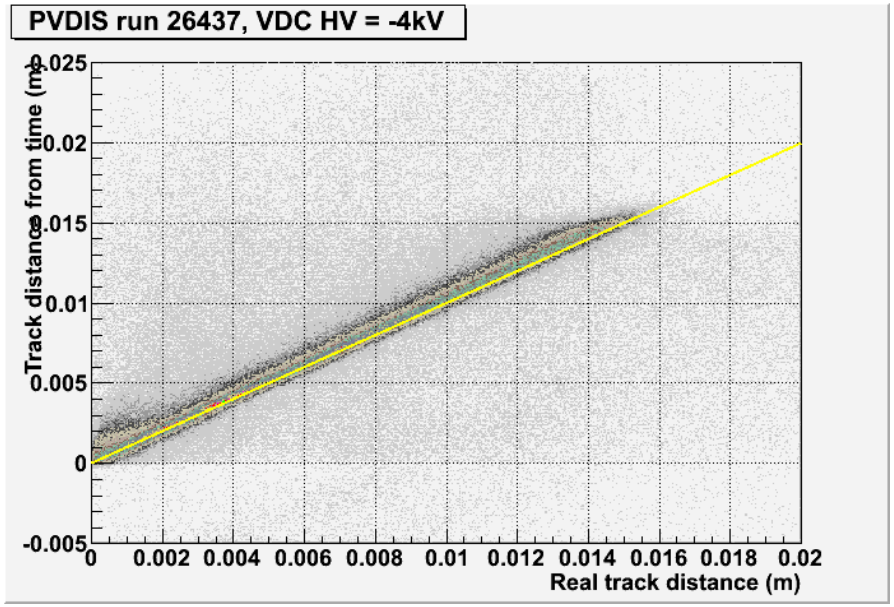
VDC time offsets





- Automated script (edge search) operating on special calibration runs (white spectrum)

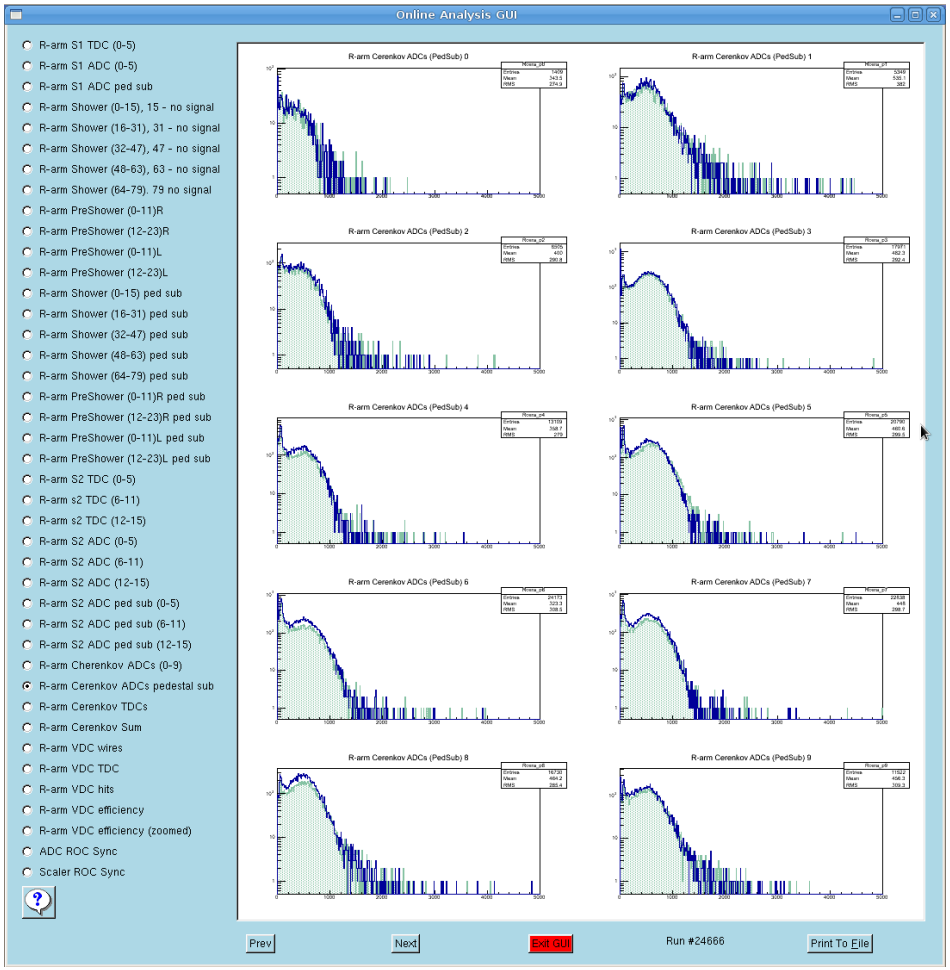
VDC time-to-distance conversion

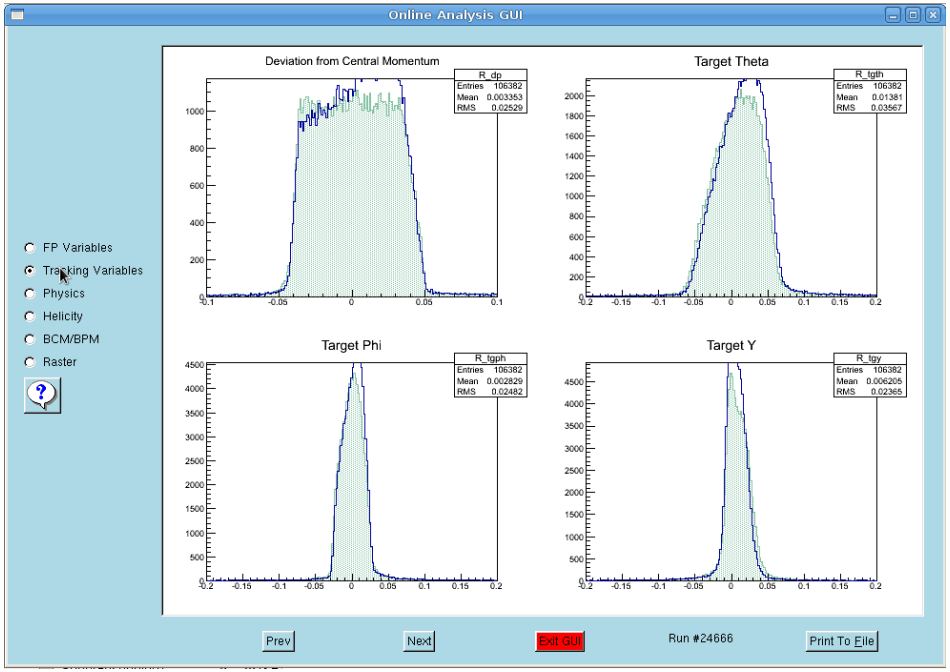


- Automated fit to analytic expression approximating time-to-distance relation
- Two linear sections with dependence on 1/tan(track angle)
- Attempts to obtain flat drift distance distribution
- Can operate on same calibration runs as time offset calibration

Online Histogramming

"OnlineGUI" (B. Moffit, 2007) ▸ doc ▸ web





- Compiled ROOT script
- Visualizes DST (ROOT) output after prompt replay
- Easy configuration via text input file
- Reference histograms (shaded)

Source Code Control & Build System

- Source code revision control
  - C++ Analyzer under CVS since 2001
  - Full file history frequently useful
  - CVS may also provide text file database history (users' choice)
  - May migrate to `git` for compatibility with Hall C
- Developer tools
  - Standard GNU build tools (`make`, g++ compiler)
  - Debugging similarly based on open source tools (`gdb`, `valgrind`)

Documentation & Developer Support

- Web-based user guide ▸ web
- `THtml`-generated reference documentation ▸ web
- Example replay scripts (from previous experiments)
- Software development kit (SDK)
  - Facilitates rapid development of new apparatus, detector, and/or physics module classes. Provides skeleton code for each module type.
  - User code is placed in a shared library (plugin) that can be loaded dynamically at run time
  - No modifications of the core analyzer code necessary

Areas of Concern, Weaknesses

- SBS software development path & milestones not well defined at this time
- HRS calibration procedures not well centralized. Certain common tasks (*e.g.* PID) tend to be re-done/re-invented by each experiment. But: Users seem to like it that way.
- Smaller issues
  - Documentation not as good as it could be, especially for beginners
  - API and database format of C++ Analyzer classes sometimes inconsistent. Could benefit from cleanup.