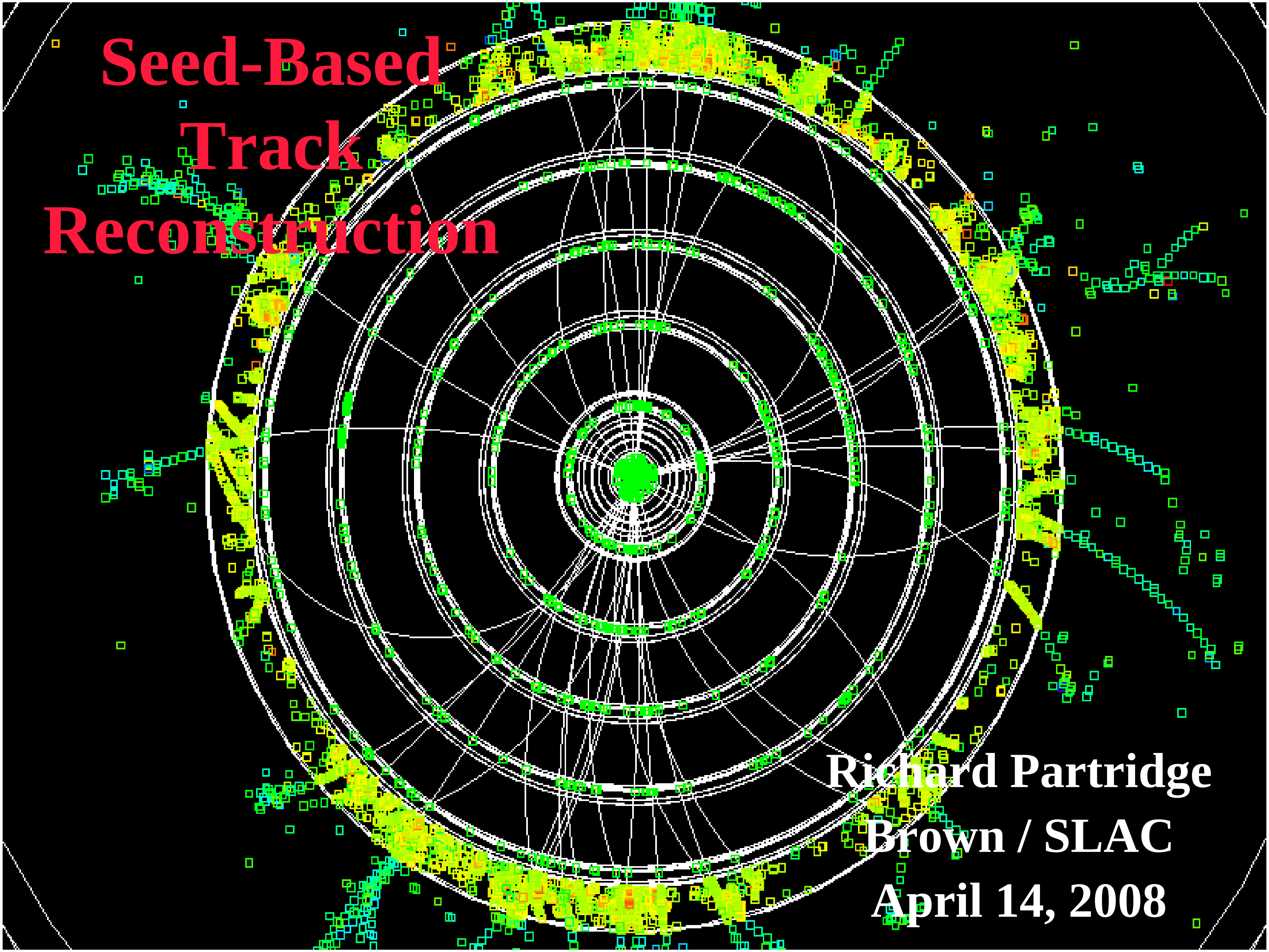
Seed-Based Track Reconstruction



Richard Partridge
Brown / SLAC
April 14, 2008

Track Reconstruction Overview

The 3 Phases of Track Reconstruction

- Hit Digitization
  - Turn GEANT energy deposits (SimTrackerHits) into hits (TrackerHits)
    - TrackerHits provide the hit position and covariance matrix used for track finding
  - Full digitization models detailed geometry in GEANT and simulates charge collection, electronic digitization, and clustering of hit strips (see Tim's talk)
  - Virtual segmentation divides cylinders / disks into virtual sensors allowing different segmentation schemes to be compared without rerunning GEANT
- Track Finding
  - Finds collections of TrackerHits that define a track
  - See Rob Kutschke's talk at the SiD workshop for a survey of SiD algorithms
  - This talk focuses on the SeedTracker package (C. Deaconu and RP)
- Track Fitting
  - Fits the track hits to determine the helix parameters and covariance matrix
  - Weight matrix fitter does a χ^2 fit including multiple scattering correlations
  - Kalman filter fitter provides adaptive fitting to mutliple scattering

SeedTracker Philosophy

- Track finding begins by forming all possible 3 hit track seeds in the three "Seed Layers" specified by the user
  - Brute force approach to finding all possible track seeds
  - Helix formed from seed hits serves as the starting point for track finding
- Track finding is guided by a set of user defined "Strategies"
  - A strategy defines layers to be used, their roles, and constraints (e.g. $p_T > x$)
- All pattern recognition code is agnostic as to the type of hit
  - No differentiation between pixel or strip, barrel or forward sensors
- A fast helix fitter, HelicalTrackFitter, plays a central role
  - This is the only piece of code that needs to understand the differences between pixels and strips, barrels and disks, etc.
- Multiple Scattering must be accounted for in track finding
  - Superb intrinsic pixel/strip resolution ⇒ MS errors will typically be dominant
- All decisions based on χ^2 from fits and constraints ($p_T > x$)
  - No internal parameters or tuning is required if tracker geometry changes

Seedtracker Algorithm

- SeedTracker undertakes the following steps:

1. Organize hits
2. Create seeds
3. Fit seeds
4. Confirm seeds
5. Extend seeds
6. Merge seeds
7. Create Tracks

- These steps are described in more detail by the slides that follow

Step 1: Organize Hits

- Before calling the SeedTracker driver, HelicalTrackHits must be created and stored in the event
  - This is best done by calling HelicalTrackHitDriver
- HelicalTrackHits are used to isolate SeedTracker from the differences in the implementations of TrackerHits
- Code currently exists to create HelicalTrackHits from both "Tim's hits" (full digitization, planar geometry) and "Dima's hits" (virtual segmentation)
- SeedTracker uses a HitManager class to organize and manage the HelicalTrackHits
  - Primarily used to allow fast retrieval of hits for a given layer
  - Layers are specified by their detector name (TrackerBarrel, VertexForward, etc.), layer number (0, 1, 2, …), and BarrelEndcapFlag (BARREL, ENDCAP_NORTH, ENDCAP_SOUTH)

Step 2: Find Seeds

- Exactly 3 layers must have been specified as being seed layers in the current strategy
- To find all possible seeds, SeedTracker loops over all viable combinations of 3 hits in the 3 seed layers
- Reduce the combinatorics by eliminating hit combinations inconsistent with p_T and impact parameter constraints
  - Iterate over all hits in the first seed layer
  - Discard hits in the second seed layer not consistent with the first layer hit and a helix having the minimum p_T and maximum impact parameter
  - Similarly, discard hits in the third seed layer not consistent with the second layer hit and a helix having the minimum p_T and maximum impact parameter
  - Further improvements are probably possible, but a completely general algorithm has so far eluded me

Step 3: Fit Seeds

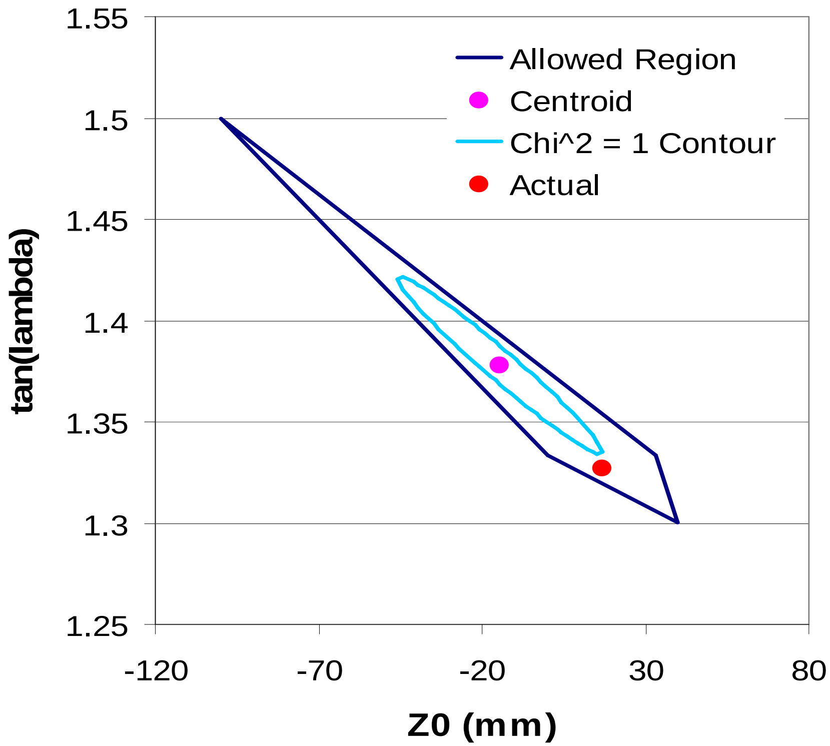
- First fit a helix to the 3 seed hits without MS errors
  - First determination of the helix parameters $\omega \equiv 1/R$, d_0, ϕ_0, z_0, and $\tan(\lambda)$
- Calculate the MS errors for each hit using this helix
- Perform a second helix fit including MS errors
- If necessary, calculate a constraint χ^2 to estimate the increase in χ^2 needed to pull into compliance with the constraint
  - Constraints: $p_T > p_T^{min}$, $|d_0| < d_0^{max}$, $|z_0| < z_0^{max}$
  - Example: if $(|z_0| > z_0^{max})$ $\chi^2 = \chi^2 + (|z0| - z_0^{max})^2 / \sigma^2(z_0)$
- Reject seeds that fail the χ^2 cut
  - χ^2 cut is applied to the sum of the fit χ^2 and the constraint χ^2

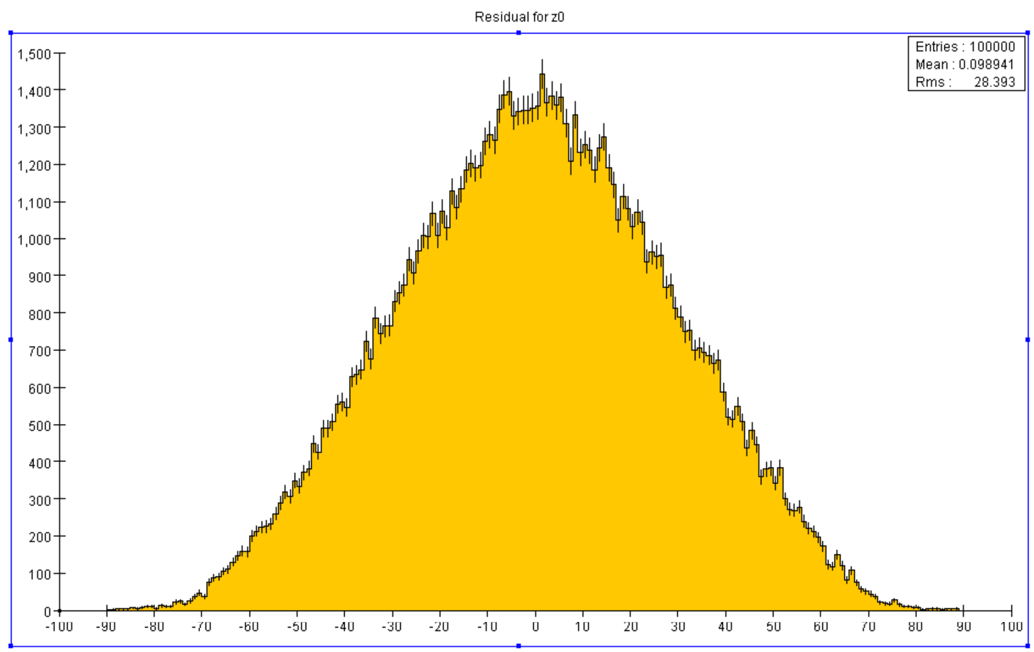
Detour: Helix Finding

- First perform a circle fit using x,y coordinates of all hits
  - Kariaki algorithm used to determines track parameters $\omega \equiv 1/R$, d_0, and ϕ_0
- Find the path lengths *s* along the track from the point of closest approach to each hit
  - Algorithm will follow curling tracks as long as the track direction changes by $<\pi$ between adjacent hits in z
- Determine the z_0 and $\tan(\lambda)$ track parameters
  - $z = z_0 + s * \tan(\lambda)$
  - If there are >1 pixel hits, do a straight-line fit using only the pixel hits
  - If there are 0 pixel hits, do a ZSegment fit using all strip hits
  - If there is 1 pixel hit, treat the pixel hit as a short strip and do a ZSegment fit

Detour: ZSegment Fit

- Strips are bounded in z ⇒ for 2 or more strip layers there are constraints on the helix paramaters z_0 and $\tan\lambda$
  - Results in a polygonal allowed region in $z_0 - \tan\lambda$ parameter space
  - "Fit parameters" are taken from centroid of allowed region
  - Covariance matrix calculated assuming all points in allowed region of parameter space are equally probable





z_0 Residual Distribution

Detour: Multiple Scattering Errors

- SeedTracker constructs a model of the tracker material
  - Each tracking element listed in the compact.xml geometry description is modeled as either a cylinder or disk
- Multiple scattering errors are calculated based on this model
  - For each hit, find all material cylinders and disks encountered between the DCA and the hit using the current estimate of the helix parameters
  - Determine the amount of material traversed in each material layer that is crossed, taking into account track angle, and calculate the MS contribution
  - Add all multiple scattering errors for this hit in quadrature
- Correlations in the MS are ignored for track finding
  - Essential for track fitting, where the accuracy of the helix error matrix is of considerable importance

Step 4: Confirm Seed

- Confirm the track seed by trying to add additional hits
  - Goal is to quickly eliminate seeds that don't correspond to real tracks
  - Hits are not necessarily required on every confirmation layer – could have 2 confirmation layers and require ≥1 confirmation hit between the 2 layers
  - One confirmation hit is typically sufficient
- Confirmation algorithm:
  - For the first confirmation layer, sort hits in this layer by their x-y distance from the helix
  - Add the closest hit to the current helix and refit
  - If an acceptable χ^2 is found, keep this hit combination
  - Loop over the sorted confirmation layer hits until the increase in the circle fit χ^2 exceeds the χ^2 cut
  - If the best fit for this confirmation layer increases the χ^2 by more than the bad hit cut, also keep the input seed
  - Repeat for any additional confirmation layers
  - Discard seeds where the minimum number of confirmation hits are not found

Step 5: Extend Seed

- Extend the seed to include hits in additional tracking layers
- Typically include all additional layers track might traverse
- Extend seed algorithm is essentially the same as the algorithm for confirming seeds
  - Strategy specifies the total number of hits required for a valid track

Step 6: Merge Seeds

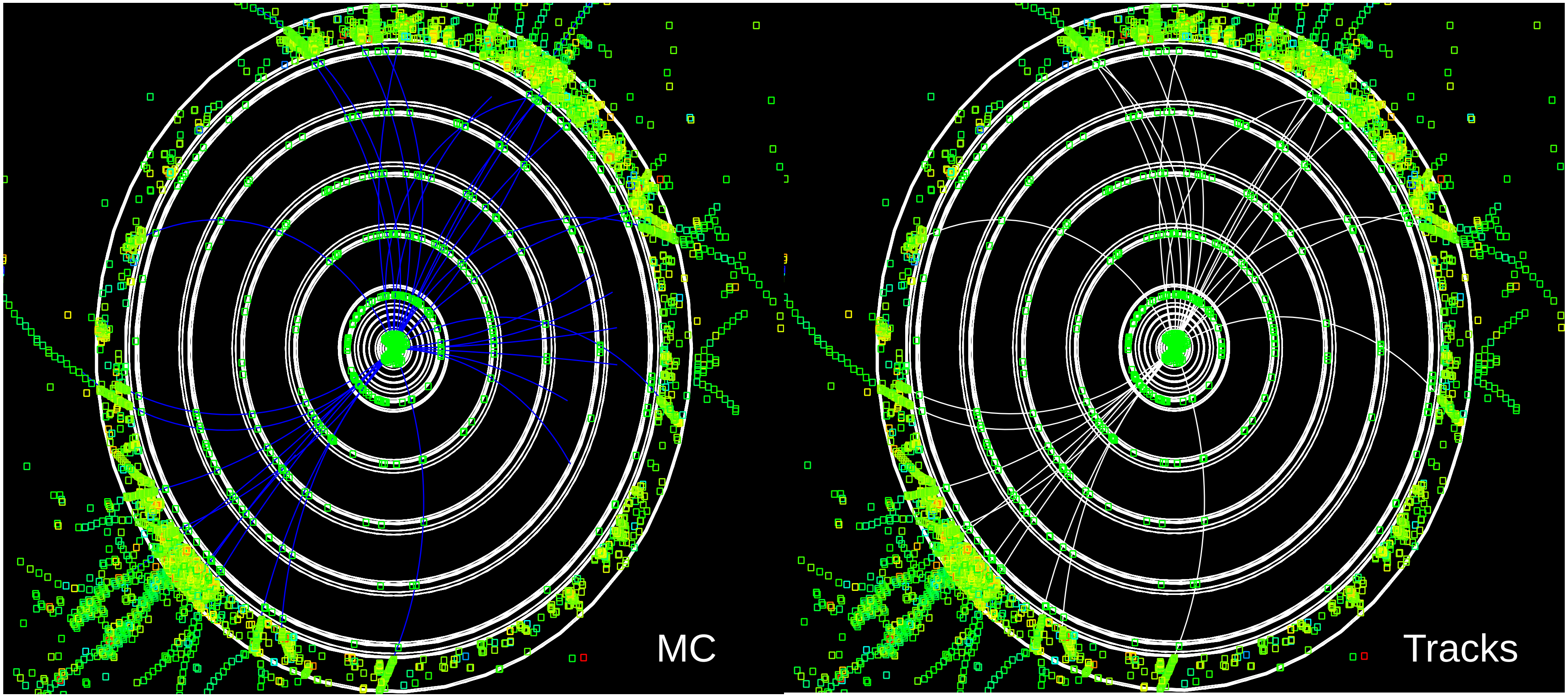
- The extend algorithm produces a collection of track seeds that satisfy all of the requirements specified by the strategy
- If multiple strategies are defined, steps 2-5 are repeated for each strategy
- There will generally be multiple similar track seeds for each real track
- Duplicate track seeds are eliminated by the merge algorithm
  - Any pair of track seeds sharing more than one hit are merged
  - If one track seed has >1 additional hit, then the seed with the most hits is kept
  - If one track seed has 1 additional hit, then the seed with the larger number of hits is kept unless the difference in χ^2 between the two track seeds exceeds the bad χ^2 cut, in which case the track seed with the better χ^2 is kept
  - If the pair of track seeds have the same number of hits, then the seed with the lower χ^2 is kept

Step 7: Create Tracks

- Tracks are created for the track seeds that survive the merge algorithm
- The track parameters and covariance matrix are taken from the last helix fit that includes all of the hits assigned to the track
- The track also contains the list of HelicalTrackHits assigned to the track seed
  - This list can then be used by track fitting algorithms

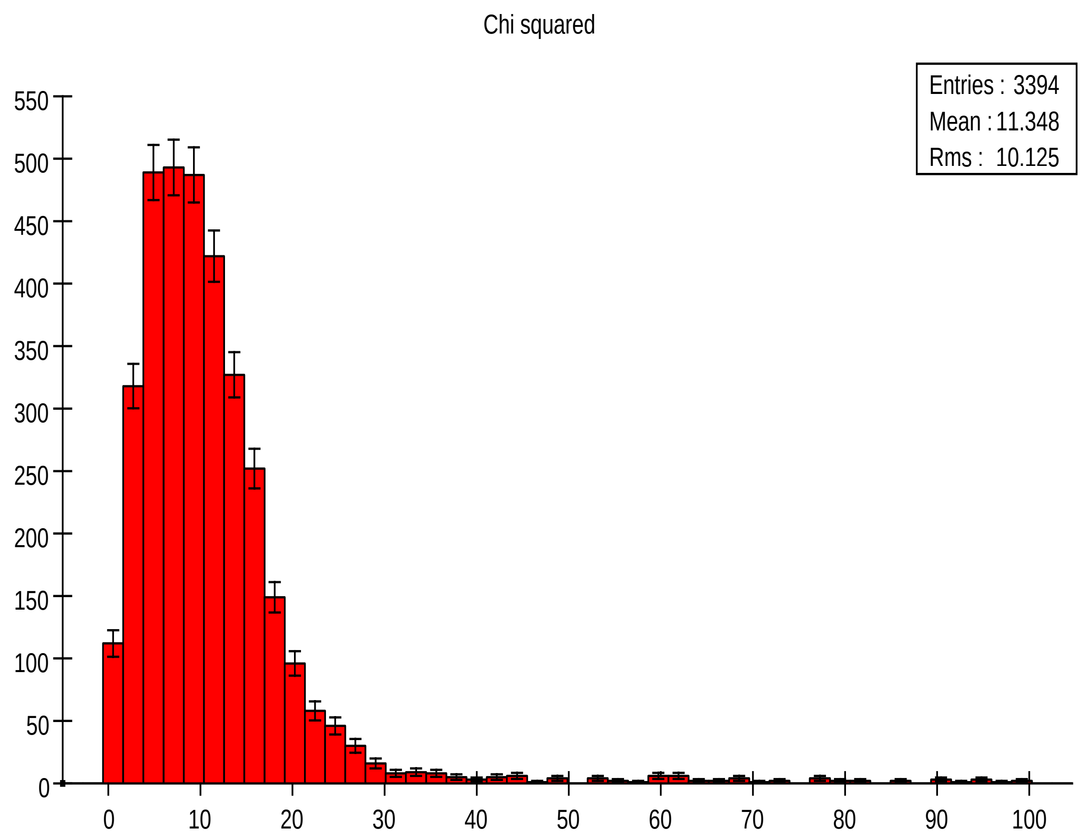
First Results

- Pythia qqbar events (uds)
- Outside in algorithm seeded from tracker barrel layers 3, 4, 5
- Inside out algorithm (vertex seeded) also works



χ^2 Distribution

- Tracks have typically 10 r-ϕ and 5 z measurements
- Helix has 5 parameters, so fits have ~10 DOF
- χ^2 distribution looks reasonable

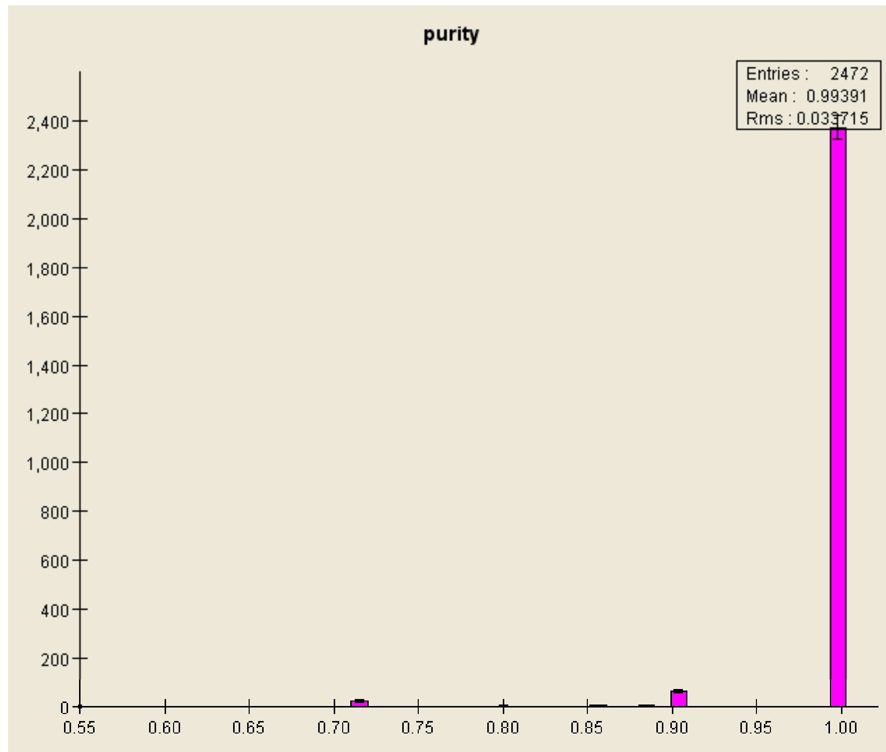


Track Purity

- Hits contains a list of MC particles that contributed to the hit
- Track purity is the fraction of hits on the track due to the MC particle with the most hits on the track
  - Purity = 1 if all hits are from the same MC particle

$$e^+e^- \rightarrow t\bar{t} \rightarrow 6 \text{ Jets}$$

Hits	Fraction	Ave Purity
7	1.3%	76.3%
8	0.2%	96.9%
9	7.7%	99.6%
10	90.9%	99.7%
All	100%	99.4%



Using SeedTracker

- The SeedTracker, HelicalTrackFitter, and ZSegmentFitter source code are in CVS at:
  - org.lcsim.contrib.seedtracker
  - org.lcsim.fit.helicaltrack
  - org.lcsim.fit.zsegment
- Code is still in active development
  - Please let Cosmin Deaconu (cozzyd@stanford.edu) and RP (richp@slac.stanford.edu) know if you are using the code so we can keep you apprised of updates / changes
- The next 3 slides show example code for setting up a strategy and an example driver for SeedTracker
  - For brevity, some preamble info was removed – the full version of this example code can be found at org.lcsim.contrib.partridge.example

Sample Code I: Setting up a Layer List

```
public class MyStrategy {
    private List<SeedStrategy> _strategylist = new ArrayList<SeedStrategy>();

    public MyStrategy() {

        // Setup the list of layers to be used and their role (seed, confirm, or extend)
        BarrelEndcapFlag barrel = BarrelEndcapFlag.BARREL;
        List<SeedLayer> lyrlist = new ArrayList<SeedLayer>();
        lyrlist.add(new SeedLayer("VertexBarrel",0,barrel,SeedType.Extend));
        lyrlist.add(new SeedLayer("VertexBarrel",1,barrel,SeedType.Extend));
        lyrlist.add(new SeedLayer("VertexBarrel",2,barrel,SeedType.Extend));
        lyrlist.add(new SeedLayer("VertexBarrel",3,barrel,SeedType.Extend));
        lyrlist.add(new SeedLayer("VertexBarrel",4,barrel,SeedType.Extend));
        lyrlist.add(new SeedLayer("TrackerBarrel",0,barrel,SeedType.Extend));
        lyrlist.add(new SeedLayer("TrackerBarrel",1,barrel,SeedType.Confirm));
        lyrlist.add(new SeedLayer("TrackerBarrel",2,barrel,SeedType.Seed));
        lyrlist.add(new SeedLayer("TrackerBarrel",3,barrel,SeedType.Seed));
        lyrlist.add(new SeedLayer("TrackerBarrel",4,barrel,SeedType.Seed));
```

Sample Code II: Creating a Strategy

```
    //  Create the "OutsideInBarrel" strategy and set its parameters
    SeedStrategy outsideinbarrel = new SeedStrategy("OutsideInBarrel",lyrlist);
    outsideinbarrel.putMinPT(1.0);           //  Set minimum pT at 1 GeV
    outsideinbarrel.putMaxDCA(1.0);       //  Set maximum d0 at 1 mm
    outsideinbarrel.putMaxZ0(1.0);          //  Set maximum z0 at 1 mm
    outsideinbarrel.putMinConfirm(1);     //  Require at least 1 confirmation hit
    outsideinbarrel.putMinHits(7);            //  Require at least 7 total hits
    outsideinbarrel.putMaxChisq(50.);      //  Set maximum chi^2 at 50
    outsideinbarrel.putBadHitChisq(15.);  //  Set chi^2 change for suspect hits

    _strategylist.add(outsideinbarrel);  //  Add the strategy to our strategy list
  }

  public List<SeedStrategy> getStrategies() {  //  Method to return the strategy list
    return _strategylist;
  }
}
```

Sample Code III: Tracking Driver

```
public class MyTrackerDriver extends Driver
{
  public MyTrackerDriver()
   {
     add(new VSExampleDriver());  //  Add a hit digitization driver

     HelicalTrackHitDriver hitdriver = new HelicalTrackHitDriver();
     hitdriver.addCollection("StandardTrackerHits",HitType.VirtualSegmentation);
     add(hitdriver);  //  Add a driver to make the HelicalTrackHits used by SeedTracker

     MyStrategy strategylist = new MyStrategy();  //  Create the SeedTracker strategies

     add(new SeedTracker(strategylist.getStrategies()));  //  Add the SeedTracker driver
   }
   public void process(EventHeader event)
   {
     super.process(event);
     return;
   }
}
```

Future Plans

- Test tracking for full digitization / planar geometry when events are available
- Finish up a few loose ends for forward tracking
- Continue to add diagnostics, tune up tracking performance, improve documentation, etc.
- Integrate track finding and track fitting
  - Nick Sinev's weight matrix fitter (χ^2 based fitter)
  - Caroline Milstein's Kalman filter fitter (developed for muon reconstruction)
  - Rob Kutschke is developing a Kalman filter fitter using the TRF packages