Tuesday

LCD Simulation Reconstruction Meeting

Rob's editorial comments in italics

Tuesday May 8, 2007 1:30 PM PDT. Ballam Conference room.

Slides available at: http://ilcagenda.linearcollider.org/conferenceDisplay.py?confId=1562

Present: Norman, Hans, Tony, Jeremy, Nick, Matt Charles, Ron Cassel, Rich, Rob, Tim, Caroline, Dima

Tim's talk:

- Tim and Jeremy wanted to avoid a big heirarchy of xml files and keep it compact.
- GeomConverter creates the big geometry from the compact xml.
  - big geometry used by both SLIC and org.lcsim

Did I get the description of the files correct?

About the multiplicity of SimTrackerHits:

When G4 propagates a track through a silicon detector, G4 may produce one or many G4 stepping points. This is under the control of various material cut parameters in G4. There was a long discussion about the optimal choices for the cut parameters: the tradeoff is speed vs fidelity. Whichever is chosen, SLIC can be configured to create one SimTrackerHit for each stepping point or it can be configured to create a single SimTrackerHit representing all of the G4 stepping points. In this latter case, the energy loss is the sum of the energy loss of summed over all G4 stepping points. I am not sure what the reported position and momentum are: but the are probably those values at one of the stepping points near the middle of the step through the material?

Can someone clarify this?

The choice of reporting one or many SimTrackerHits per track/detector pair is set in the compact geometry file but, at present, that information is not propagated so that it is accessable during reconstruction. It should be. In practice, with existing files, one may infer this information from the getPathLength() method of each SimTrackerHit, and if necessary, search for additional hits. In principle one must search the entire SimTrackerHits container.

Nick currently uses very small steps in his pixel code. He has files that were generated with 5 micron steps but not smaller. Nick has not yet implmented with straggling in energy loss but he plans to.

Norman asked if the cylindrical geometry could be represented using this scheme. I don't remember the answer - does anyone?

IDDecoder

There was a long discussion on whether not the IDdecoder should be something that gets returned by the RawTrackerHit or something else. The argument in favour of this is that users are guaranteed to get the correct IDDecoder. If they have to just know the correct IDDecoder, they will get the wrong one.

The present IDDecoder was mostly driven by the calorimeter people (since they were out of the gate first?). At one time there were separate IDDecoders for calorimeters and trackers but some time ago they were merged into a single IDDecoder. We can undo that and define a tracking only IDDecoder if that makes sense.

If we move to separate IDDecoders, persist the objects, and read them back in, then how does the reader know which IDDecoder to attach to the hits? The answer needs to be provided in the meta data for the container that is stored in the event header. This is currently done for the calorimeter.

In the cvs head version of Jeremy's code, the IDDecoder will correctly parse the fields in the in the ID's that it finds and the methods that return field values will be correct. The rest of the functionality, in particular getPosition() is not yet working.

Many in the room argued that getPosition() is not a function that should be supported by and ID decoder because a position cannot be determined in a meaningful way by:

1. a strip device
2. a pixel device with clock to end of column readout
3. a TPC hit
   In cases 2 and 3 one can return a meaningful postion if one knows the correct beam crossing to use as time0. A strip device on its own can never return a meaningful position.

Not everyone agreed with this point of view and argued that there are ways to hack a position in all cases (assume midpoint of the unmeasured coordinates).

I do think that there was agreement that on moving to a tracking specific IDdecoder.

Jeremy is working on an IDHelper that will provide useful functionality for parsing the ID's in the new geometry system. It does not implement the IDDecoder interface.

The ID's as they are currently conceived in the new geometry system are not ID's for the entire detector, just for the tracking/vertexing systems.

Norman asked: if we go with the plans above, what happens if we want the Marlin people to read our simulated events. There was not a clear answer on this: some people felt that is was not important for the Marlin people to be able to read hit level data but others did. We all agreed that it was importand for them to be able to read our tracks.

Some general principles that were suggested at various times:

- We should separate objects that change from event to event from objects that are static throughout the job. For example geometry is static throughout the job and hits change from event to event.
- If the new geometry is designed to replace the old geometry, then we should remove from the API functionality that is not implemented in the new system.
    - This may imply maintenance of code that used the old API.
- It would be good to be able to read event data without requiring that the geometry be instantiated. Of course there would be things that could not be done unless the geometry is instantiated.
    - One solution is for the geometry to be instantiated on demand, as can be done with a singleton pattern.

Question/Work List:

1. Propagate the setting of flag that controls the number of SimTrackerHits per track/sensor intersection so that it is accessible during reconstruction.
2. Fix the bug in RawTrackerHit so that it returns a list of SimTrackerHits, not just a single one.
3. Make a decision on the IDdecoder issues this week and implement it soonafterwards.
4. What does it mean to have multiple events in memory and how is that handled? It is meaningful to merge events at any stage of the processing chain. (If we want hits with high fidelity, it is important to make the hits before pulse heights are digitized.)

Matt Charles' talk

Found a Bug in his Code

In his code he has a variable named doca for distance of closest approach. It is not the distance of closest approach of a track to a line. Instead it is the pathlength along the track from the current place to the point of closest approach (to z axis, to vertex?). There was a bug in this computation. When it was fixed, the jet energy resolution did not improve as much as expected.

Investigating the Poor Resolutions

The jet energy resolution estimated by simulation remain much poorer than expected, even when he used a pure cheater to perform all of the tracking. He investigated and discovered that the problem was the following. Their code propagates tracks with the MC true parameters from the IP to the calorimeter face using an exact helix. For low momentum tracks this algorithm may miss the true impact point by many cm (due to scattering and energy loss). Similarly when a track decays in flight, this calculation will give an incorrect answer. When this extrapolation misses the true impact point on the calorimeter by too much, then the track is not correctly matched to its calorimeter this and the PFA has poor energy resolution.

Matt's solution was to use the cheater to find the correct last 3 hits on the track, fit a helix to these hits and extrapolate it to the calorimeter. This improved things a great deal. There remain a small number of outliers that are due:

- The last three hits are a long distance from the calorimeter (decay in flight ?)
- Bugs in correctly identifying the correct last three hits

In the followup discussion we agreed on part of the general solution to this situation. The reconstructed track class should provide the track parameters plus covariance matrices at many different points along the track, one of which should be at the outermost hit. The implementation should allow the parameters to be stored at an arbitary number of places. When track parameters are returned they should include information that says they are valid for some range of path lengths, relative to the reported point.

It is a detail for later to decide whether the standard tracking code should also extrapolate from the last hit to the face of the calorimeter or if that job should be left to the calorimeter code.

If the tracks are not correctly matched to clusters, shouldn't this systematically overestimate the jet energy since the track energy is counted twice? I remember that the resolution functions were fat but not biased? Did I misremember or is there another explanation?

The Fermilab MCFast algorithm would be the ideal solution to this problem: Cheat to find all of the hits on all of the tracks; for tracks with enough hits, kalman filter them both directions and store the track parameters at many places. This can be implemented using one of the existing kalman filters or perhaps with Nick's weight matrix fitter.

Rob's list of N places at which to store the track parameters + covariance matrix:

- *Point of closest approach to the z axis.*
- *Point of closest approach to MC truth production vertex.*
    - *Very useful for resolution studies.*
    - *Only needed if this point is not within the beampipe vacuum.*
- *Innnermost hit.*
- *Outermost hit.*
- *Impact point on the face of the ecal or on some well defined surface close to the ECal.*
  I would store all 5 by default and provide methods to drop unneeded parameter sets to save disk space, when they are not needed. I would also provide the option to store the full parameters at all measurement and scattering surfaces.