Summary of Summer 08 Simulation Work

Cosmin Deaconu

Stanford / SLAC

August 29, 2008

Full support for digitized hits in SeedTracker

- SeedTracker can use fully digitized hits (Nick Sinev's PixSim + Tim Nelson's SiStripSim) in addition to hits from Virtual Segmentation.
- Both the cylindrical and planar geometries are supported, including material modeling (which is important for calculating multiple scattering errors). This was the hard part.
- Virtual Segmentation is faster, but full digitization is (hopefully!) more realistic.
- The CVS version of the planar geometry still has a bug in it with two sensitive vertex barrel layers. While this shouldn't affect the results of tracking, it does slow things down quite a bit.

Digitization performance

Table: Some unscientific benchmarking done with JAS3 on my laptop w/ a 1.86 GHz Pentium M (2 MB L2 cache)

	pythiaZPoleuds-0		
	1st event (w/ setup)	100 events	Final memory
VSeg	12.42 s	503.987 s	11.1 MB
Digi	42.27 s	837.92 s	395MB

	panpyttbar-0		
	1st event (w/ setup)	50 events	Final memory
VSeg	21.07 s	???	<50 MB
Digi	47.78 s	1083.24 s	438 MB

NB: Things seem to go significantly slower when invoked by mvn exec:java, especially with the digitized hits. No idea why.

Additional notes on performance of hit digitization

- PixSim and SiStripSim take approximately the same time to process an event. (PixSim uses table lookup for some calculations though.)
- While the physics events should be approximately the same for both geometries, there are a few differences.
  - The strategy list for the planar tracker is longer (23 strategies vs. 21 strategies). This is partially a reflection of hit making not being completely efficient with digitization, so the strategies tend to be more redundant.
  - Because of overlaps with the planar geometry, there are more hits in the tracker barrel (probably not too many more, but SeedTracker run time scales as some polynomial...)
  - Lots of geometry info for the planar tracker $\rightarrow$ lots of memory use $\rightarrow$ perhaps more garbage collection overhead

Strategy Builder Overview

- Initially, there were inefficiencies with SeedTracker in the planar geometry. This is because the strategies were simply ported over from sid01. The difficulty of writing strategies by hand motivated me to create an automatic strategy builder.
- Results have been pretty good: in sid01 ZPole events, Rich's handcrafted strategies had 98.5 efficiency with "findable" tracks (not curlers, θ not too small). Strategies generated by Strategy Builder for sid01 seem to get over 99.5 percent efficiency in ZPole events.
- As a byproduct of the work on StrategyBuilder, we now have an XML exchange format for lists of strategies (schema and all).
- StrategyBuilder is a driver which can either be run through JAS3 or with the main class in RunStrategyBuilder
- For input, an event and a layer weights file should be provided (though a default is defined for both sid01 and sid01_planar_tracker that will be used if nothing else is given). Optionally, a prototype strategy or a starting set of strategies may also be used.

Strategy Builder Algorithm (Process Step)

- The process phase of the driver is used to gather information. Two key pieces of data are compiled using SimTrackerHits:
    1. The number of particles that pass through each layer set of size [seed layers + confirm layers].
    2. How often any given layer set is found to be adjacent.
- Particles may be filtered out before information gathering. By default, a filter using the cutoff parameters of the prototype strategy is used, but any class implementing IFilter may be used.
- Because hit digitization is not perfectly efficient, assuming 100 percent hit making efficiency during this step leads to missed tracks. For this reason, the weights file can specify a default efficiency as well as per SimTrackerHit collection efficiencies. Efficiencies less than 1.0 will cause SimTrackerHits to be randomly ignored with the appropriate probability.
- The weights file also specifies whether or not the layer number must be divided by two in the tracker endcap / forward (this is because every weights file has a target detector).

Strategy Builder Algorithm (Scoring Step)

- After we have gathered all the necessary data, we want to pick a set of strategies that is both fairly short and providing good coverage.
- The way this is done is by trying every possible layer set and choosing the maximally scoring set until sufficient coverage is achieved.
- The score is calculated in the following way:

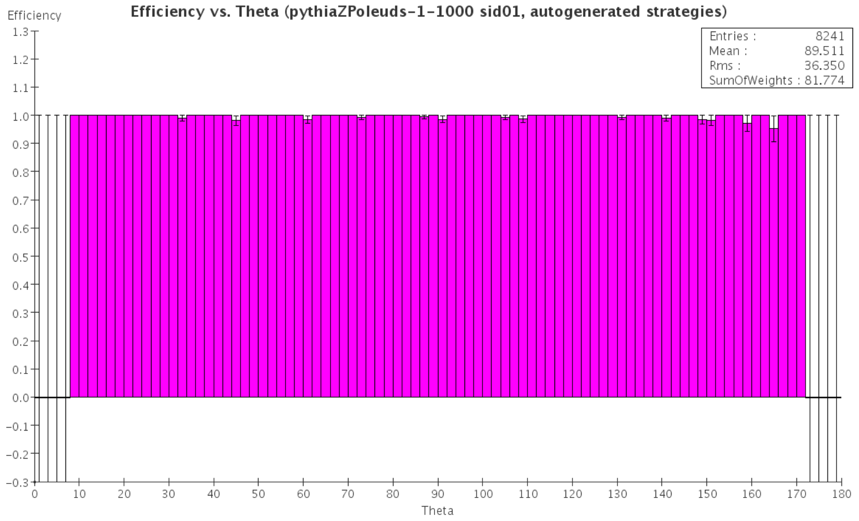
  $$Score = N \cdot W \cdot (c + m \cdot A^e) \tag{1}$$

  Where N is the number of additional tracks that could possibly be found by the set (i.e. the particle contains hits in all layers), W is the average layer weight of the layers in the set (specified in the weights file), A is the number of the times that the layers in the subset were found to be adjacent and the other values are constants specified in the weights file (I've been using $c = 2$, $m = 1$ and $e = 0.1$).
- One can specify the minimum value of N before breaking out of the scoring loop (if one doesn't want to generate a whole bunch of strategies that can only find one additional particle, for example).

Strategy Builder Algorithm (Strategy generation step)

- To find the extend layers associated with a given set, we look at all other layers hit by particles which go through all the layers in a set.
- Disambiguation between seed and confirm layers is done as follows: (this part of the code is kind of rough right now)
  - If all the layers are adjacent, then either the first or last *nconfirm* are chosen for confirmation depending on the layer weight of the first or last hit (lower = confirm).
  - If at least *nseed* layers are adjacent, then those are chosen for seeding and the rest for confirm.
  - If neither of these is the case, then the confirm layers are the ones with the lowest layer weights.
- If the symmetrize option is enabled (it is by default) then all generated strategies will be force symmetrized between the north and south endcap.
- An XML file will be output with the final strategy list.

Strategy Builder Result



NaN hunting

- SeedTracker used to contain a lot of bugs that came from calculations involving NaN values. NaN's are somewhat pernicious because of the way the propagate through calculations. It can take a long time to figure out the cause.
- I *think* that those have been fixed now except for one outstanding bug in the CircleFitter code.
    - There was a case where a circle was being fit where two points had the same x and y coordinates (though different z coordinates), which can happen at a non-negligible frequency with Virtual Segmentation. Such input to to the CircleFitter will currently cause NaN output.

Persistence

- To help ameliorate the effect of long digitization time, one approach is to save an lcio file with the hits already digitized (obviously this only works if you're using the same data more than once).
- Until recently, this was impossible due to a number of bugs.
- I've tracked down those bugs and it is now possible to persist Nick's hits and Tim's strip hits. Tim's cross hits cannot be persisted right now, but SeedTracker ignores those anyway.
- In org.lcsim.recon.tracking.seedtracker.digiexample can be found a DigiHitWriter and DigiHitReader class which can be used to write and read PixSim-generated and SiTrackerHitStrip1D hits. The reader also converts to HelicalTrackHits which can readily be used for tracking.
- Tracks output from SeedTracker can be persisted (obviously).
    - Caveat: Hits from Virtual Segmentation currently don't seem to have any way to get the list of RawTrackerHits required to create proper TrackerHits, so the list of TrackerHits associated with a Track is not 100 percent well-formed when using Virtual Segmentation.

The End

Thank you all for a great summer!