ILCD simulation efforts at NIU

Dhiman Chakraborty
NIU/NICADD

International Linear Collider
Large Detector Meeting
LLR, Paris, January 13–15, 2005
Tasks

- Full-detector simulation:
  LCDG4 (for ALCPG)
- Test-beam module simulation:
  TBMokka (with DESY/LLR, for CALICE)
- Simulation of detector imperfections:
  DigiSim (for ALCPG, CALICE, …)
Full detector simulation: LCDG4

- GEANT4 v6.2 (v7.0 coming soon).
- Most hadronic physics lists available, LCPhysics not tested yet.
- Input format: binary STDHEP
- Output format: .lcio (standard) or .sio (Gismo compatible)
- Nominal American detector geometries are implemented via XML geometry files (continuous tubes + disks only).
LCDG4: some nice features

- Correct MC particle hierarchy, even when $V^0$s and hyperon decays are forced at the event generation stage.
- Energies deposited in absorbers are available for cross-checks.
- Non-projective geometries available.
- Simple analysis code and documentation available from CVS and http://nicadd.niu.edu/~lima/lcdg4/.
LCDG4: Projective vs. non-projective

- Barrel only for now.
- Rectangular virtual cells with linear dimensions, controlled at run time (XML).
- Ecal and HCal can be made projective or non-projective independently.
**LCDG4: XML geometry description**

**Example:** barrel HCal (dimensions in cm).

```xml
...<volume id="HAD_BARREL" rad_len_cm="1.133" inter_len_cm="0.1193">
  <tube>
    <barrel_dimensions inner_r="144.0" outer_z="286.0" />
    <layering n="34">
      <slice material="Stainless_steel" width="2.0" />
      <slice material="Polystyrene" width="1.0" sensitive="yes" />
    </layering>
    <!--segmentation cos_theta = "600" phi = "1200" /-->
    <cell_info length="1.0" height="1.0" offset="0." outer_r="246.5"/>
  </tube>
  <calorimeter type="had" />
</volume>
...
```

Most of detector changes are very easy to make!

Proj or NP selection either one, not both.
LCDG4 output: general features

- One particle collection and several hit collections (one hit collection per subdetector)
- Each hit points to the contributing particles (except tracker hits from calorimeter back-scatters).
  - LCIO only: (x,y,z) coordinates stored for every hit
- All secondaries above an energy threshold (now set at 1 MeV), except for shower secondaries, are saved in output with:
  - Particle id and generation/simulation status codes.
  - Production momentum, production* and ending position (*LCIO only).
  - Calorimeter entrance: position and momentum (SIO only).
  - Pointers to parent particles.
Decays forced in generator correctly processed

$\Xi^0 \rightarrow \Lambda^0 \pi^0$
LCDG4: example: $e^+e^- \rightarrow tt$
LCDG4: wrap-up

- Presently the ALCPG standard (until SLIC is ready for production).
- Code with doc + instructions available: http://nicadd.niu.edu/~lima/lcdg4
- Many interesting single particle and full event samples available, new ones can be requested:

For detailed access instructions, see http://nicadd.niu.edu/~jeremy/admin/scp/index.html
DigiSim: A simulation package to help simulate various detector effects for the ILC

Outline

- Purpose and requirements
- Package description
- Usage and configuration
- Developing new functionality
- Test results
- Current status and summary
Introduction

- Goal: a program to simulate the signal collection and digitization process in the ILC detector(s).
- Converts the GEANT4 output (energy depositions and space–time coordinates) into the same format AND as close as possible to real data from readout channels.
- Same code can then be used to process MC and real data.
- Serves a convenient and standardized hook for the user to model such detector effects as inefficiencies, non-uniformities, non–linearities, attenuation, noise, cross-talk, hot and dead channels, cell ganging, etc. involved in signal collection, propagation, and conversion/recording.
Requirements and choices

- Basic requirements:
  - Object-oriented design to simplify maintenance and implementation of new functionality
  - Should be used within the CALICE test beam project, as a testbed for the reconstruction framework
  - All test beam code based on C++ and LCIO
  - Marlin (v0.6) chosen as the C++ framework
LCIO event model

SimCalorimeterHit
   _cellID : int
   _energy : float
   _mcpvec : MCPartContVec

RawCalorimeterHit
   _channelID : int
   _amplitude : int
   _timeStamp : int

CalorimeterHit
   _cellID : int
   _energy : float
   _time : float
   _type : int
   _rawHit : LCOobject*

MCParticleCont
Particle : MCParticle*
Energy : float
Time : float
PDG : int

LCEvent

LCCollection

LCCollection

SimCalorimeterHits --> RawCalorimeterHits
Reco: RawCalorimeterHits --> CalorimeterHits
LCRelations associating SimHits to RawHits have not yet been implemented
**DigiSim class diagrams**

- **MARLIN**
  - Processor
    - CalHitMapProcessor
      - CalHitMapMgr
        - Utility for optimal data access (singleton)
    - DigiSimProcessor
      - CalHitModifier
        - FunctionModifier
        - GainDiscrimination
      - TempCalHit
        - Gain + threshold
    - SmearedLinear
      - CalDigiCrossTalk
      - Function-based transformations

- Marlin framework
  - Transient class used by modifiers

ILC-ILD meeting, 15-01-2005

ILCD simulation efforts at NIU
Dhiman Chakraborty
Calorimeter hits are shown here, but it applies just as well to other subdetectors.

TempCalHits are both input and output to each modifier.

All processing is controlled by a DigiSimProcessor (one per subdetector).

Modifiers are configured at run time, via the Marlin steering file.

New modifiers can be easily created for new functionality (more info later).
Steering file example

# Example steering file for DigiSim

.begin Global

# specify one or more input files (in one or more lists)
LCIOInputFiles inputfile

# the active processors that are called in the given list
ActiveProcessors CalHitMapProcessor
ActiveProcessors EMDigiSimProcessor
ActiveProcessors HADDigiSimProcessor
ActiveProcessors TCDigiSimProcessor
ActiveProcessors OutputProcessor

# limit the number of processed records (run+evt):
MaxRecordNumber 500
.end Global
**Configuring processors and modifiers**

```
# A DigiSim processor. It instantiates one or more calorimeter hit 
# "modifiers", which together represent the full digitization process
.begin EMDigiSimProcessor
ProcessorType       DigiSimProcessor
InputCollection     EMcalCollection
OutputCollection    EMrawCollection

ModifierNames       EMDigiIdent EMGainThresh EMThreshOnly EMFixedGain

# Parameters:    type               gainNom gainSig  thresh   thrSig
EMFixedGain        GainDiscrimination 1000000   0    0       0
EMThreshOnly       GainDiscrimination    1      0      30      0
EMGainThresh       GainDiscrimination    1000000 50000  30      1.5

# A function-based modifier     ElinNom ElinSig TlinNom  TlinSig
EMDigiIdent        SmearedLinear       1      0      1      0
.end
```

(As many as needed)
GainDiscrimination is a smeared lin. transf. + threshold on energy

SmearedLinear is a func-based smeared linear transformation on energy and/or timing

Example modifiers
Creating new modifiers

CalHitModifier

- `setDebug(int)`: void
- `newInstance()`: CalHitModifier* = 0
- `init(floats)`: void = 0
- `processHits(TempCalHits)`: void=0
- `print()`: void = 0

FunctionModifier

- `par`: floats
- `init(floats)`: void
- `processHits(TempCalHits)`: void
- `print()`: void
- `transformEnergy(hit)`: double = 0
- `transformTime(hit)`: double = 0

GainDiscrimination

- `_par`: floats
- `newInstance()`: CalHitModifier*
- `init(floats)`: void
- `processHits(TempCalHits)`: void
- `print()`: void
- `energyToADC(double)`: double
- `isBelowThreshold(double)`: bool

SmearedLinear

- `newInstance()`: SmearedLinear*
- `transformEnergy(hit)`: double
- `transformTime(hit)`: double
- `_par`: floats

YourClassHere

- `_par`: floats
- `newInstance()`: CalHitModifier*
- `init(floats)`: void
- `processHits(TempCalHits)`: void
- `print()`: void
- `energyToADC(double)`: double
- `isBelowThreshold(double)`: bool

Start by copying either one of the existing classes, then implement what is needed.
10 GeV $\pi^\pm$ on ECAL: gain + threshold

Comparing EM ADC counts - 10 GeV pions

- Fixed gain, no threshold (FGNT)
- Gaussian gain, no threshold (GGNT)
- Gaussian gain, fixed threshold (GGFT)
- Gaussian gain, Gaussian threshold (GGGT)

“gain” = 1000000
threshold = 125

$\sim$140 keV 140 counts
same plot as before, now with log-x

10 GeV $\pi^\pm$ on ECal: gain + threshold
10 GeV $\pi^\pm$ on ECal: time stamps

No time smearing in this example
10 GeV $\pi^\pm$ on HCAl: gain + threshold

$\sim 1.7$MeV 1700 ADC counts

“gain” = 1000000
threshold = 1750
10 GeV $\pi^\pm$ on HCal: gain + threshold

Comparing HAD energies - 10 GeV pions - SDJan03

- Fixed gain - No threshold
- Gauss gain - No threshold
- Gauss gain - Fixed threshold
- Gauss gain - Gauss threshold

<table>
<thead>
<tr>
<th>Fixed gain - No threshold</th>
<th>Entries: 445370</th>
<th>Mean: 2.3577</th>
<th>Rms: 1.0073</th>
<th>OUTORange: 4901</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gauss gain - No threshold</td>
<td>Entries: 442395</td>
<td>Mean: 2.3730</td>
<td>Rms: 0.99210</td>
<td>OUTORange: 7876</td>
</tr>
<tr>
<td>Gauss gain - Fixed threshold</td>
<td>Entries: 117262</td>
<td>Mean: 3.5707</td>
<td>Rms: 0.28421</td>
<td>OUTORange: 4</td>
</tr>
<tr>
<td>Gauss gain - Gauss threshold</td>
<td>Entries: 116448</td>
<td>Mean: 3.5646</td>
<td>Rms: 0.29429</td>
<td>OUTORange: 4</td>
</tr>
</tbody>
</table>

ILC LD meeting, 15-01-2005
ILCD simulation efforts at NIU
Dhiman Chakraborty
10 GeV $\pi^\pm$ on HCal: time stamps

Comparing HAD time stamps - 10 GeV pions

- Fixed gain - No threshold
  - Entries: 426171
  - Mean: 6.7637
  - Rms: 4.5271
  - OutOfRange: 24100

- Gauss gain - No threshold
  - Entries: 426171
  - Mean: 6.7637
  - Rms: 4.5271
  - OutOfRange: 24100

- Gauss gain - Fixed threshold
  - Entries: 315430
  - Mean: 6.3936
  - Rms: 3.4894
  - OutOfRange: 1857

- Gauss gain - Gauss threshold
  - Entries: 314600
  - Mean: 6.1262
  - Rms: 3.5827
  - OutOfRange: 1851
Status

- A first version of DigiSim (proof of concept) is implemented
- Two real modifiers implemented:
  - GainDiscrimination and function-based SmearedLinear
- LCRelation coming (soon?), example code at hand (thanks to F. Gaede)
- Output LCIO files contain ECal and HCal RawCalorimeterHit collections, while keeping simulation collections untouched.
- Analysis code developed for examining raw hits, plots confirm expected behavior.
- Creation of new (simple) modifiers is easy: just copy one of the existing modifier classes and implementing the desired transformation.
- Some complicated ones: crosstalk and cell ganging require neighborhood information, from geometry-aware classes. NonProj code exists (java and C++), but projective geometry only available in java.
Summary

- A first version of a digitization simulation program, DigiSim, has been developed at NIU for ILC full detector and test–beam programs.
- DigiSim is object–oriented, powerful, extensible, yet very simple implementation using C++
- Digitization of tracking detectors can probably be implemented as easily as the calorimeter hits.
- CALICE test beam can use DigiSim as a testbed, for evaluation and improvement suggestions. Please use it for your digitization studies.
- Documentation including download and building instructions available at [http://nicadd.niu.edu/digisim/DigiSim.html](http://nicadd.niu.edu/digisim/DigiSim.html),
- User feedback is urgently sought. Send any questions or comments to: lima@nicadd.niu.edu and dhiman@fnal.gov.