A digitization simulation package for the International Linear Collider

Guilherme Lima
for the ILC group at NIU

LC Simulations Workshop
DESY, December 10, 2004
Talk Outline

- Singletons!
- **DigiSim**: Purpose and requirements
- Package description
- Usage and configuration
- Developing new functionality
- Preliminary results
- Current status and summary
A typical clustering code

```cpp
LCCollection* col = _event->getCollection("EMcalCollection");

SimCalorimeterHit *ihit, *jhit;
for( int i = 0; i < col->getNumberOfElements(); ++i ) { // loop over EM hits
    ihit = (SimCalorimeterHit*)col->getElementAt(i);
    int icellID = ihit->getCellID0();

    // find a neighbor hit
    int neighID = findANeighborHit(icellID);

    for( int j = 0; j < col->getNumberOfElements(); ++j ) { // a nested loop!
        jhit = (SimCalorimeterHit*)col->getElementAt(j);
        int jcellID = jhit.getCellID0();

        if( jcellID == neighID ) { // neighbor hit found
            // ...do some processing here...
        }
    }
}
```
Better: use hit maps

LCCollection* col = _event->getCollection(“EMcalCollection”);

CalHitMap hitmap;                                   // Create a hit map
for( int i = 0; i<col->getNumberOfElements(); ++i ) {
  ihit = (SimCalorimeterHit*)col->getElementAt(i);
  int cellID = ihit->getCellID0();
  hitmap[cellID] = ihit;                              // populate hit map
}

SimCalorimeterHit *ihit, *jhit;
CalHitMap::iterator iter, jter;
for( iter = hitmap.begin(); iter != hitmap.end(); ++iter) {  // loop over hits
  ihit = *iter;
  int icellID = ihit->getCellID0();

  int neighID = findANeighborHit(icellID);            // find a neighbor
  jter = find( hitmap.begin(); hitmap.end(), neighID );
  if( jter!=hitmap.end() ) {                           // neighbor found
    jhit = *jter;
    // ...do some processing here...
  }
}
Even better: use a singleton

```cpp
// fetch the hit map filled by a singleton class
CalHitMapMgr* pHitMgr = CalHitMapMgr::getInstance();
CalHitMap& hitmap = pHitMgr->getCollHitMap("EMcalCollection");

for( int i = 0; i<col->getNumberOfElements(); ++i ) {
    ihit = (SimCalorimeterHit*)col->getElementAt(i);
    int cellID = ihit->getCellID0();

    // look for a neighbor
    int neigbID = getANeighbor();
    jhit = find( hitMap.begin(); hitMap.end(); neigbID );
    //... do some processing here ...
}
```

Remember: people out there are copying your code into their analysis/reconstruction code. It is up to you to give them good code to be copied!
Uh? What's a singleton?!

- Singleton: a class behavior to enforce that only a single instance of a given class will ever exist.
- Example: optimize data access (hit maps keyed by cellID)
- Other hit collections can also be made available in a similar optimized way

```cpp
typedef map<int, SimCaloHit*> CalHitMap;

class CalHitMapMgr {
public:
    // To be used instead of constructor
    static CalHitMapMgr* getInstance() {
        if (!_me) _me = new CalHitMapMgr();
        return _me;
    }

    static void destroy() {if(_me) delete _me;}

    CalHitMap& getCollHitMap(string& name);

private:
    CalHitMapMgr(); // constructor
    ~CalHitMapMgr(); // destructor

private:
    static CalHitMapMgr* _me;
};
```
DigiSim

- Goal: a program to simulate the digitization process for the ILC detector simulation

- DigiSim role is to convert the simulated data (energy depositions and hit timings) into the same format AND *as close as possible* to real data from readout channels.

- Same reco / analysis software can be used for MC and real data.

- *As close as possible* means that all known effects from digitization process should be taken into account, if possible: cell ganging, inefficiencies, noise, crosstalks, hot and dead channels, non-uniformities, etc.
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

• Gaede's Marlin (v0.6) chosen as the C++ framework
LCIO event model

**LCEvent**

- **SimCalorimeterHit**
  - _cellID : int
  - _energy : float
  - _mcpvec : MPartContVec

- **RawCalorimeterHit**
  - _channelID : int
  - _amplitude : int
  - _timeStamp : int

- **CalorimeterHit**
  - _cellID : int
  - _energy : float
  - _time : float
  - _type : int
  - _rawHit : LCOBJECT*

**LCRelation**

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

**MARLIN**

*Processor*

1

**CalHitMapProcessor**

1

**CalHitMapMgr**

1

**DigiSimProcessor**

*CalHitModifier*

1

**TempCalHit**

Gain + threshold

Utility for optimal data access (singleton)

Function-based transformations

Marlin framework

Transient class used by modifiers

SmearedLinear
Calorimeter hits are shown here, but the same structure could also be used for tracker hits

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

# the active processors that are called in the given order
ActiveProcessors CalHitMapProcessor
ActiveProcessors EMDigiSimProcessor
ActiveProcessors HADDDigiSimProcessor
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

(As many as needed)

ModifierNames     EMFixedGain  EMThreshOnly  EMDigiIdent

# 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  ---------------------------------------------------------------
Existing modifiers

GainDiscrimination is a smeared lin. transf. + threshold on energy

SmearedLinear is a func-based smeared linear transformation on energy and/or timing
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* = 0
- init(floats) : void
- processHits(TempCalHits) : void
- print() : void
- energyToADC(double) : double
- isBelowThreshold(double) : bool

SmearedLinear
- newInstance() : SmearedLinear* = 0
- transformEnergy(hit) : double
- transformTime(hit) : double

YourClassHere

Start by copying either one of the existing classes, then implement what is needed
10 GeV ± on EM: gain + threshold

~140keV ⇒ 140 counts

“gain” = 1000000
threshold = 125

Fixed gain, no threshold (FGNT)
Gaussian gain, no threshold (GGNT)
Gaussian gain, fixed threshold (GGFT)
Gaussian gain, gaussian threshold (GGGT)
10 GeV ± on EM: gain + threshold

same plot as before, now with log-x
10 GeV ± on EM: time stamps

No time smearing in this example
10 GeV ± on HAD: gain + threshold

~1.7MeV ⇒ 1700 ADC counts

“gain” = 1000000
threshold = 1750
10 GeV ± on HAD: gain + threshold

FGNT
GGNT
GGFT
GGGT
10 GeV \pm on HAD: time stamps
Status

- A first version of DigiSim (proof of concept) is implemented
  - Two real modifiers implemented: GainDiscrimination and function-based SmearedLinear
  - LCRelation: to be used (soon?), example code at hand (thanks to F.Gaede)
- Output LCIO files contain EM and HAD RawCalorimeterHit collections, while keeping simulation collections untouched.
- Analysis code for plotting raw hits, plots confirm expected behavior.
- Creation of new (simple) modifiers is quite easy, by just copying 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, aiming at full digitization of ILC detector simulations.
- 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.
- Send any questions or comments to me: lima at nicadd.niu.edu
- Documentation available at http://nicadd.niu.edu/digisim/DigiSim.html, including download and building instructions.
Hier heute zu sein ist nett.
Ich hofft bald zurückzukommen!

Danke!

Disclaimer: grammatical errors are google's, not mine!
10 GeV ± on EM: time stamps

Comparing EM time stamps – 10 GeV pions

- Fixed gain – No threshold
  - Entries: 202229
  - Mean: 0.66681
  - Rms: 0.31737
- Gauss gain – No threshold
  - Entries: 202229
  - Mean: 0.66681
  - Rms: 0.31737
- Gauss gain – Fixed threshold
  - Entries: 155357
  - Mean: 0.66175
  - Rms: 0.30845
- Gauss gain – Gauss threshold
  - Entries: 155357
  - Mean: 0.66111
  - Rms: 0.30663

x ~ 1, + log scale = binning effect

Same plot as before, now in log scale
10 GeV ± on HAD: time stamps

Comparing HAD time stamps – 10 GeV pions – SDJan03

- Fixed gain – No threshold
  - Entries: 450,265
  - Mean: 0.88796
  - Rms: 0.45512
  - OutOfRange: 6

- Gauss gain – No threshold
  - Entries: 450,265
  - Mean: 0.88796
  - Rms: 0.45512
  - OutOfRange: 6

- Gauss gain – Fixed threshold
  - Entries: 117,287
  - Mean: 0.80861
  - Rms: 0.29871

- Gauss gain – Gauss threshold
  - Entries: 116,450
  - Mean: 0.81192
  - Rms: 0.29984
  - OutOfRange: 1