



Implementation of the BeamCal Electron Tagging Algorithm

André Sailer

CERN-PH-LCD

FCal Collaboration Meeting
Cracow, April 30, 2013

Table of Contents



1 BeamCal Detector Layout

2 Clustering Algorithm

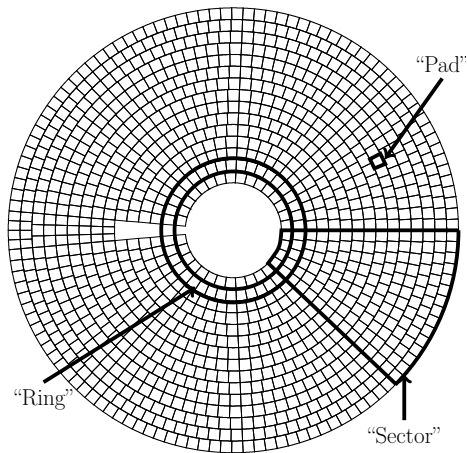
3 Implementation Details

4 Application Example

BeamCal Detector at CLIC



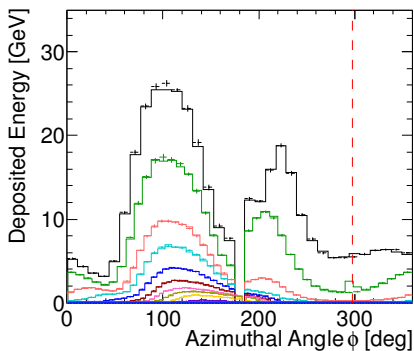
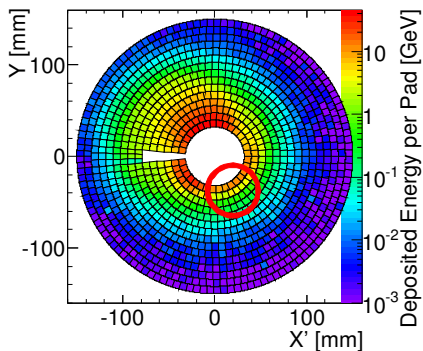
- 10 mrad to 45 mrad (ILC about 3.5 mrad)
- Absorber for incoherent pairs
- Mask for downstream elements (QD0, BPM)
- Radiation hard sensors required
- Electron tagging for background suppression
- Tungsten sandwich calorimeter, Molière radius of about 1 cm
- Pad size $8 \times 8 \text{ mm}^2$
- ILC opening around outgoing beam pipe larger, different number of pads per sector



High Energy Electron and Background



- 1.5 TeV Electron on top of 40 BX background
- Have to identify cells and cluster with more energy deposited than average background



BeamCal Electron Clustering Algorithm



General Idea of Clustering

- 1 Energy in the BeamCal Pads is

$$\vec{E}_{\text{Total}}^{\text{Event}} = \vec{E}_{\text{Signal}}^{\text{Event}} + \vec{E}_{\text{BKG}}^{\text{Event}}$$

- 2 Subtract the average background plus one standard deviation for each cell

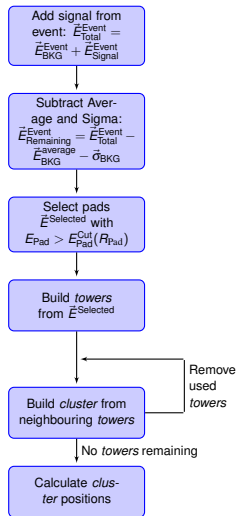
$$\vec{E}_{\text{Remaining}}^{\text{Event}} = \vec{E}_{\text{Total}}^{\text{Event}} - (\vec{E}_{\text{BKG}}^{\text{average}} + \vec{\sigma}_{\text{BKG}})$$

- 3 Select the pads with sufficiently large remaining energy, e.g., $E_{\text{Remaining}}^i > \sigma_{\text{BKG}}^i$

- 4 Find *towers* in the selected pads: Pads with the same position in different layers

- 5 Merge neighbouring *towers* into *cluster*

- 6 Calculate *cluster* position



Problems with Existing Implementation



- Existing MARLIN Processor too tightly coupled with ILD BeamCal (hardcoded numbers everywhere)
- Some not so nice features in the code (copying 3D arrays by hand everywhere)
- Wanted to use electron tagging for CLIC BeamCal: Different number of layers, radii, cutout angles, etc.
- Re-implemented algorithm more generally

The Implementation in PseudoCode



- On the next slides the abridged implementation is shown in c++ PseudoCode
- Is nonetheless rather technical
- Showing the **types**, **variables**, **important functions**, and **geometry functions**
- Based on three classes
 - ▶ **BCPadEnergies**: A vector of the energies per Pad, plus functionality to manipulate, add, subtract, scale, cluster pads
 - ▶ **BCPadCuts**: A class to implement pad and cluster selection
 - ▶ **BeamCalGeo**: Interface to **Geometry** information (Gear, DD4hep, ...)
- Not showing event display and other code
- Not showing `const(&)`, iterators, etc.
instead of

```
for(std::vector<int>::iterator it = container.begin();  
it != container.end(); ++it)
```

write

```
forall ( int number in container ){ } //not valid c++
```

Function: doClustering



```
BeamCalClusterList doClustering(BCPadEnergies totalEnergy ,
                                   BCPadEnergies background ,
                                   BCPadCuts cuts ) {
    BeamCalClusterList beamCalClusters;
    totalEnergy.subtractEnergies(background);

    PadIndexList selectedPads = getPadsAboveSigma( totalEnergy , backgroundSigma , cuts );

    while( not selectedPads.empty() ) {
        PadIndexList padsForNextRound;
        TowerIndexList myTowerIndices = getTowersFromPads( selectedPads );
        TowerIndex largestTower = max_element( myTowerIndices );
        forall( TowerIndex testTower in myTowerIndices ) {
            if ( not areNeighbours( largestTower , testTower ) ) {
                selectedPads.removeTower( testTower );
                padsForNextRound.addTower( testTower );
            }
        }
        beamCalClusters.addCluster( getClusterFromAcceptedPads( totalEnergy , selectedPads ) );
        //try again for the remaining pads without the cluster that was already found
        selectedPads = padsForNextRound;
    }//while there are towers

    return beamCalClusters;
}
```


Function: subtractEnergies



```
void BCPadEnergies::subtractEnergies(BCPadEnergies testPads){
    for (PadIndex globalPadID = 0; globalPadID < getNumberOfPadsPerBeamCal();
         ++globalPadID) {
        this->m_PadEnergies[globalPadID] -= testPads.m_PadEnergies[globalPadID] ;
    }
}
```

- Same idea for addition, scaling, etc.
- Everything is based on the vector of energy per pad
- Can be vectorised easily for parallelisation

Function: getPadsAboveSigma



```
PadIndexList getPadsAboveSigma(BCPadEnergies testPads ,
                                BCPadEnergies backgroundSigma ,
                                BCPadCuts cuts) {
    PadIndexList selectedPads;
    for (int globalPadID = 0; globalPadID < getNumberOfPadsPerBeamCal(); ++globalPadID ) {
        if ( testPads.getLayer( globalPadID ) >= cuts.getStartingLayer() ) {
            double padEnergy = testPads.getEnergy( globalPadID );
            double padSigma = backgroundSigma.getEnergy( globalPadID );
            if ( padEnergy > cuts.getPadSigmaCut() * padSigma ) {
                selectedPads.addPad( globalPadID );
            }
        }
    }
    return selectedPads;
}
```

- This functions returns a list of the selected pads (integers)
- Only the globalPadID is stored
- Energy of the pads can be found from the existing objects

Function: getTowersFromPads



```
TowerIndexList BCPadEnergies::getTowersFromPads( PadIndexList selectedPads ) {
    TowerIndexList myTowerIndices;
    forall ( PadIndex globalPadID in selectedPads ) {
        myTowerIndices[ globalPadID % getPadsPerLayer() ] += 1;
    }
    return myTowerIndices;
}
```

- Returns the a list of *Towers*
- The index gives the `globalPadID`
- The value is the number of pads in the *Tower* (could also be the sum of the energy in the tower)

Function: areNeighbours



```
bool areNeighbours( int globalPadID1, int globalPadID2 ) {
    int layerID1, layerID2, ringID1, ringID2, padID1, padID2;
    getLayerRingPad( globalPadID1, layerID1, ringID1, padID1 );
    getLayerRingPad( globalPadID2, layerID2, ringID2, padID2 );
    if( ringID1 == ringID2 ) {
        if( //The index of neighbouring pads differs by 1
            ( abs( padID1 - padID2 ) <= 1 ) or
            // or one is zero and the other is getSegmentsPerRing(ringID1) - 1
            ( ringID1 >= getFirstFullRing() and
              ( ( abs( padID1 - padID2 ) + 1 ) == getSegmentsPerRing(ringID1) ) ) )
        ) { return true; } else { return false; }
    }
    if( abs( ringID1 - ringID2 ) == 1 ) {
        double phi1 = getPadPhiDegree( ringID1, padID1 );
        double phi2 = getPadPhiDegree( ringID2, padID2 );
        if( fabs( phi1 - phi2 ) < 18.0 ) {
            return true; } else { return false; }
        }
    }
    return false;
}
```

- Checking if two pads have adjacent indices
- Azimuthal angle check is a bit untidy, could be improved I think
- Took out the special case where the pads have to be in the same layer

Function: getClusterFromAcceptedPads



```
BeamCalCluster getClusterFromAcceptedPads(BCPadEnergies testPads ,  
                                             PadIndexList selectedPads ) {  
    BeamCalCluster cluster ;  
    double phiAverage(0.0), ringAverage(0.0), totalEnergy(0.0);  
    double sinStore(0.0), cosStore(0.0); // for average phi  
    forall ( PadIndex globalPadID in selectedPads ) {  
        double energy( testPads.getEnergy( globalPadID ) );  
        cluster.addPad( globalPadID , energy );  
        totalEnergy += energy ;  
        ringAverage += getRing( globalPadID ) * energy ;  
        sinStore += energy * sin( getPadPhiRad( globalPadID ) );  
        cosStore += energy * cos( getPadPhiRad( globalPadID ) );  
    }  
    if( totalEnergy > 0.0 ) {  
        phiAverage = atan2( sinStore/totalEnergy , cosStore/totalEnergy );  
        ringAverage /= totalEnergy ;  
        cluster.setPhi( phiAverage );  
        cluster.setTheta( getThetaFromRing( ringAverage ) );  
    }  
    return cluster ;  
}
```

- Take the list of selected pads
- Calculate the energy weighted position of the cluster
- Finished

Function: doClustering



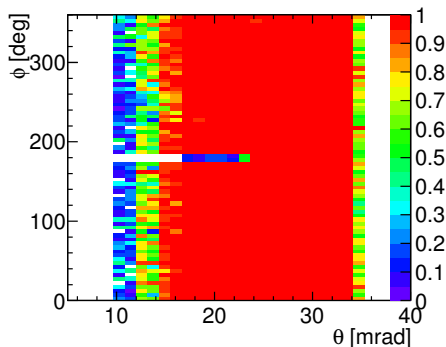
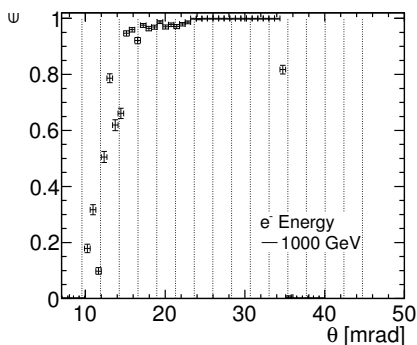
```
BeamCalClusterList doClustering(BCPadEnergies totalEnergy ,
                                   BCPadEnergies background ,
                                   BCPadCuts cuts ) {
    BeamCalClusterList beamCalClusters;
    totalEnergy.subtractEnergies(background);

    PadIndexList selectedPads = getPadsAboveSigma( totalEnergy , backgroundSigma , cuts );

    while( not selectedPads.empty() ) {
        PadIndexList padsForNextRound;
        TowerIndexList myTowerIndices = getTowersFromPads( selectedPads );
        TowerIndex largestTower = max_element( myTowerIndices );
        forall( TowerIndex testTower in myTowerIndices ) {
            if ( not areNeighbours( largestTower , testTower ) ) {
                selectedPads.removeTower( testTower );
                padsForNextRound.addTower( testTower );
            }
        }
        beamCalClusters.addCluster( getClusterFromAcceptedPads( totalEnergy , selectedPads ) );
        //try again for the remaining pads without the cluster that was already found
        selectedPads = padsForNextRound;
    }//while there are towers

    return beamCalClusters;
}
```

- Used algorithm to study efficiency of Electron Tagging at 3 TeV (reported at last year's meeting at Desy)
- Example efficiencies for 1 TeV electrons with 40 BX of background





- Implemented BeamCal Electron Tagging algorithm
- Connect reconstruction to DD4hep geometry interface as a use-case
- Create LCI0 output format, so it can be used in MARLIN reconstruction and analysis chain



Backup Slides

Pad ID Functions



```
void getLayerRingPad( int globalPadID , int& layer , int& ring , int& pad ) {  
    //layer starts at 1!  
    layer = getLayer( globalPadID );  
    ring = getRing ( globalPadID );  
    int ringIndex = globalPadID % getPadsPerLayer();  
    pad = ringIndex - getPadsBeforeRing( ring );  
    return;  
}  
  
int getLayer( int globalPadID ) {  
    //layer starts at 1  
    return ( globalPadID / getPadsPerLayer() ) + 1;  
}  
  
int getRing( int globalPadID ) {  
    element = std::upper_bound( getListPadsBeforeRing().begin()+1,  
                               getListPadsBeforeRing().end(),  
                               globalPadID % getPadsPerLayer() );  
    return element - m_PadsBeforeRing.begin() - 1 ;  
}
```