Documentation of the DeepAnalysis code

Vasiliy Morgunov
edited by: Erika Garutti

November 30, 2007

1 Introduction

The hadron shower DeepAnalysis code has been developed with the propose of studying the composition of hadronic showers utilizing data from highly granular calorimeter prototypes. It is an ad hoc type of clustering algorithm based on analog information of energy deposited in a hadronic shower tuned for the identification of the shower components. This multi-purposes algorithm make a decomposition of hadronic shower (set of calorimeter hits) into a number of clusters with different physical properties.
As a warning from the author, this code is not tuned for showers separation and it is strongly geometry dependent. It is a FORTRAN code written at the beginning of 2003 with aim of design of hadron calorimeter prototype. The DeepAnalysis was first applied in a series of studies all available in http://www.desy.de/∼morgunov
"Test-Beam Prototype Volume Coverage and Clusters in It" (2003)
"Tile Size Issues for HCAL Prototype" (2003)
"Prototype Geometry influence on Reconstruction Quality" (2003)
"Attempt to estimate the prediction power of the calorimeter properties following from different hadron models in the simulation programs; and the systematic errors of such predictions" "Two Particle Separation with Tile HCAL" (2004) by A. Raspereza.
Recently algorithm was rewritten in C++ by A. Zhelezov. The rewriting was done in two steps: first the code was rewritten first in C to reproduce the exact output numbers for each event with the computer accuracy, then the C version was rewritten in C++ to get a compact code.

2 The code concept

The main idea of the DeepAnalysis code is to separate a hadronic shower in four contributions: track-like particles (TRK), hadronic fraction (HAD), electromagnetic fraction (EM) and neutrons (NEU). The DeepAnalysis proceeds sequentially through

the three basic steps: hit classification, hit clustering and joining of clusters. The hits are classified according to their energy into three types: TRK, HAD and EM like. Fig. 1 a) shows the energy spectrum of single hits and an example of energy cuts to identify the types. These energy boundaries are parameters of the algorithm, normally set to 0.5-1.7 mips (TRK-like), 1.7-3.5 mips (HAD-like) and larger than 3.5 mips (EM-like). An additional hit class is introduced for NEU like hits, which is not based on energy information and it will be discussed after clustering. After hit classification a two-dimensional clustering is performed in each calorimeter layer. Subsequently, clusters in consecutive layers are joined in a three-dimensional procedure using their topological properties. The TRK like clusters are classified as having large eccentricity and low hit density. The HAD like clusters have relatively small eccentricity and low hit density. The EM like clusters have high hit density and large eccentricity. The EM and the HAD clusters have to be connected by TRK clusters. The remaining isolated hits not assigned during this procedure are regarded as NEU like hits. These hits are by definition disconnected from the shower tree but they get assigned to the shower if their distance from the shower axis is less than a defined parameter. See Marius Groll thesis (http://www-flc.desy.de/flc/internal/paper/thesis.2007.groll.pdf) thesis for a first example of application of this code to calorimeter data.

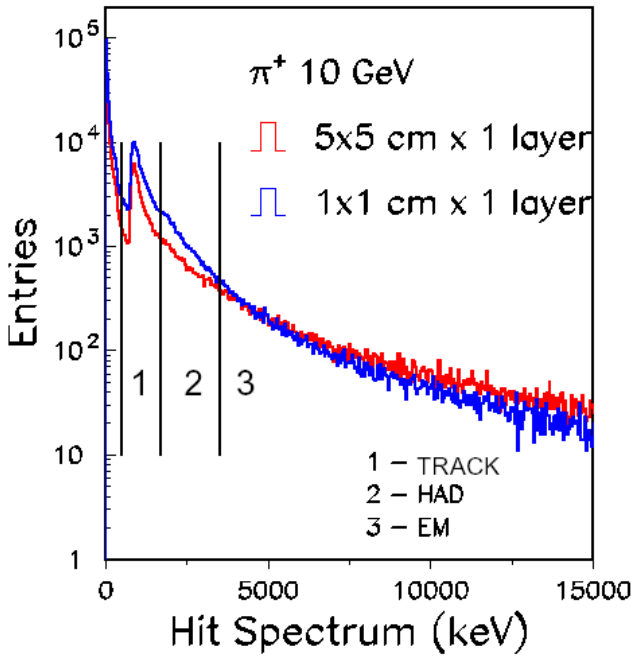

Figure 1: *Hit class classification by hit energy.*

3 Input / Output

The DeepAnalysis code accepts hits as input and returns clusters as output. The code requires from the main user program a list of hits initially tagged to start the reconstruction. Hits are given to the program using the mothod:

```
deep_analysis.add_hit(x,y,z,ampl_gev,ampl_mip,layer,type);
```

where `x,y,z` are the space coordinates of the hit[1]; `ampl_gev (ampl_mip)` is the energy amplitude of one hit expressed in GeV (number of MIPs); and `type` is a pre-assigned hit type. The definition of hit type is part of the code user. An example is shown where three threshold are used to separate noise, track-like, hadron-like and electromagnetic-like hits.

```
if(ampl_m < threshold_0)
continue;
else if(ampl_m < threshold_1)
type = DeepAnalysis::TRK;
else if(ampl_m < threshold_2)
type = DeepAnalysis::HAD;
else
type = DeepAnalysis::EM;
```

The thresholds are one set of the important parameters to tune in the code. An example of thresholds used by the code author is given:

```
// These numbers in MIPs are the color boundary definition
// The colors are defined as 0 < C1 < C2 < C3 < C4
// COLOR = no color – just a noise
float threshold_0 = 0.50;
// Hit color = 3 = green − − − > Track like
float threshold_1 = 2.0;
// Hit color = 4 = blue − − − > Hadronic like
//float threshold_2 = 4.0;
float threshold_2 = 4.5;
// Hit color = 2 = red − − − > Electromagmetic like
```

[1]The DeepAnalisys coordinate system is centered at the center of hte ECAL front plate. The positive x-axis pointing to the right, the positive y-axis pointing up and the z-axis closing the right-handed coordinate system points upstream the beam direction.

It has been checked that the code is very sensitive to the choice of threshold_0 since this threshold directly influences the total number of noise hits which decreases esponentially with threshold. Important is also the choice of threshold_2 since (how it will become clear in the following) the code starts to claster from the electromagnetic core. The distinction between track-like and hadron-like hits has very little influence on the code output and threshold_1 can be also set equal to threshold_2 with no strong impact.
Note these thresholds are in principle energy dependent and the user could consider implementing variable thresholds as a function of the beam energy.

Once the input hits are given the DeepAnalysis code is called and the statistics of the output clusters can be check using a print method for each cluster type.

```
deep_analysis.reconstruction();
deep_analysis.color_clusters_print_stat(DeepAnalysis::EM);
deep_analysis.color_clusters_print_stat(DeepAnalysis::TRK);
deep_analysis.color_clusters_print_stat(DeepAnalysis::HAD);
deep_analysis.color_clusters_print_stat(DeepAnalysis::NEUTR);
```

The method deep_analysis.color_clusters_print_stat() returns the energy sum of all clusters, the total number of hits and number of clusters for a given type. The same information can also be stored into variables using the method:

```
deep_analysis.color_clusters_stat(type,Energy,hits,clusters);
```

4 The code parameters

The DeepAnalysis works with a handful of parameters which optimize the clustering for a specific detector geometry. The list reported below has been tuned for the CALICE AHCAL prototype detector (http://www-flc.desy.de/hcal/tilehcal/). Using the code for any other detector geometry would require full retuning of all parameters (including the energy thresholds reported in the previous section).

```
Detector()
sampling = 30.0;
cell_size = 30.0;
ext_cell_size = 30.0;
neut_thresh = 0.6;
normal_thresh = 0.5;
elips_trans_to_catch= 1.5;
elips_forw_to_catch = 1.5;
```

elips_back_to_catch = 1.0;
ecc_trk_hadr_sel = 0.3;
a_delta = 0.01;
sph_cut = 0.05; // about 3 degree
cut_3d_em = 31.0;
cut_3d_had = 81.0;
join_factor = 1.0;
zoom_dist = 200.0;
The meaning of each parameter is explained in the following when addressing the code in details.

5 The DeepAnalysis code

The main routine of the DeepAnalysis code is reported here below:

```
    //---------------------------------------------------
void reconstruction() // Main routine
//---------------------------------------------------
// cout<<" NO CLUSTER JOINING "<<endl;
all_hits- >calc_stat();
detector.init(all_hits- >r_g.min);
event_move();

    // cout<<" ---------- EM like ----------"<<endl;
cluster_finder_3d(*(kind[EM]),false);
cluster_join(true);
em_shower_find();
reassignment_small_clusters(TRK);

    // cout<<" ---------- TRK like ----------"<<endl;
cluster_finder_3d(*(kind[TRK]),false);
cluster_join(false);
reassignment_small_clusters(HAD);

    // cout<<" ---------- HAD like ----------"<<endl;
cluster_finder_3d(*(kind[HAD]),true);
// cluster_join(false);
// low_e_clean();
event_move_back();
for(RSIterator<Cluster> ci(clusters);ci.next();)
ci- >finalize(*this);
```

The code is organized in three main blocks dealing with EM, TRK and HAD-like clusters in sequence. The order of the various steps is relevant and should not be mixed. Before and after the block for cluster handling the event is rotated fro and back away from the z-axis. The rotation matrix used is:

```
z = x+zoom_dist
x = z
y = -y
```

This rotation is needed to avoid the singularity between Θ=0^o and Θ=90^o, when working in polar coordinates.
To follow a brief description of the main DeepAnalisys routines in the order of call.

5.0.1 `cluster_finder_3d`

The 3D cluster finder proceeds in steps, first the detector volume is divided in spherical shells. Each shell is treated as a 2D plane in spherical coordinates and a 2D cluster finder is applied. Then 2D clusters on various shells are merged to form 3D objects.
The definition of shells is made according to the sketch in Fig. 2 and depends on three

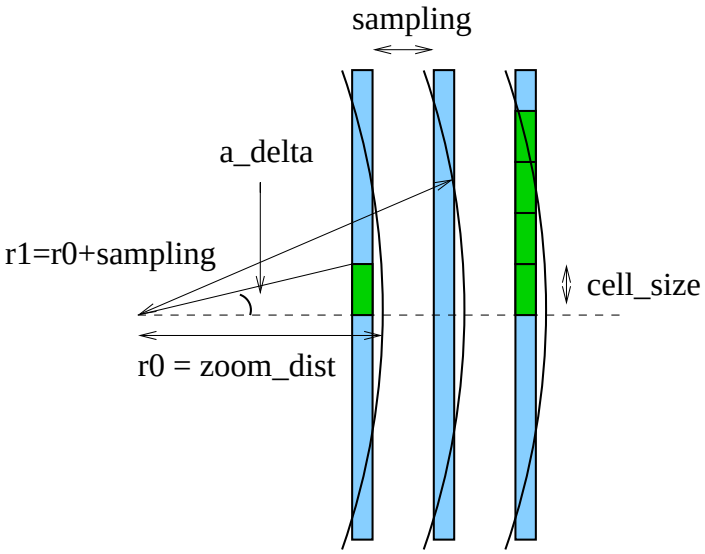

Figure 2: *Sketch of geometry definition of shells in 3D clustering procedure.*

user parameters. The parameter `zoom_dist` [mm] defines the radius of the first shell. Ideally the shell center should be adjusted according to the position where the shower starts. This would require setting `zoom_dist` event by event. A simpler but less precise solution is to fix the center of the first shell about 200 mm in front of the first detector layer. The parameter `sampling` [mm] defines the distance between shells. `cell_size`

defines the granularity of a grid on the 2D plane and should be set similar to the detector granularity. The parameter `a_delta` is defined as the ratio `cell_size/zoom_dist` $= tan(\Theta) \sim \Theta$.
Once shells are defined a 2D clustering is applied which works in the (Θ, ϕ) plane. A 2D histogram is filled with binning $r \sim$ `a_delta`. On the histogram projection (see Fig. 3)

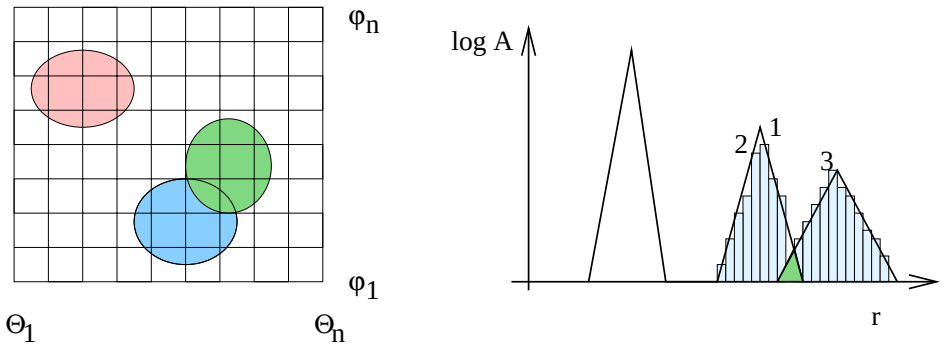

Figure 3: *Sketch of 2D clustering procedure.*

a peak finder based on bin content amplitude is used to identify the center of a cluster (1) and associate the histogram bins direct neighbors to the peak (2) till a second peak (3) is found with no direct neighbor to cluster 1. This procedure continues till overlap is found between two clusters. The common histogram bins are randomly assigned to one of the two clusters with a weighted probability. Note that changing `a_delta` will change the number of bins in the 2D histogram and therefore will impact on the number of 2D clusters found. The radius of action of the peak finder is steered by the `sph_cut` parameter. Increasing the value of `sph_cut` will decrease the number of 2D clusters found.

After the 2D clustering is compleated on each shell the routine `cluster_find` is called. The distances in 3D between the clusters are calculated, then starting from the closest clusters a chain is formed between all 2D clusters with distance $d <$ `R_join` (see Fig 4). At the end of one chain a second is started from the next minimum distance between the remaining 2D clusters. The parameter `R_join = dr * join_factor` can be tuned by the user changing the value of `join_factor`. A larger `join_factor` value will merge more 2D clusters to one chain.
For reference, the value of `dr` is calculated for each 2D cluster, give the layer (or shell number) where the 2D cluster is found and its radius, as: $dr = \frac{radius}{layer}\left(1 + \frac{1}{layer}\right)$.

5.0.2 `cluster_join`

This routine is used to join nearby 3D clusters with distance smaller that the `cut_3d_em` (`cut_3d_had`) for EM (TRK or HAD) like clusters. For analysis of hadronic showers it

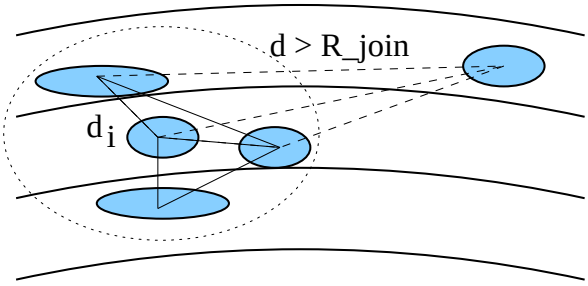

Figure 4: *Sketch of 2D cluster chaining procedure in 3D.*

is suggested to use `cluster_join` only to join EM like clusters, while the HAD option can be useful in muon analysis to avoid the splitting of a muon track. In this routine clusters with one or two hits in the surroundings of an EM cluster are merged. This helps collect the tails of electromagnetic shower around the high energy core.

5.0.3 `em_shower_find`

For EM showers another step is necessary to include all hits on the tails of the shower. For this purpose an ellipsoid is built around the EM shower core, which is defined by three parameters: a transverse radius and a backward and forward radius (to account for the forward boost of the shower). The three radii of the ellipsoid are then increased by the values of the parameters `elips_trans_to_catch`, `elips_forw_to_catch`, `elips_back_to_catch`.
The tuning of this three parameters is essential to control the electromagnetic fraction identified by the code. The best method for tuning is to compare in MC true π^0 energy in a hadronic shower with EM reconstructed energy. This tuning has been performed using the CALICE AHCAL prototype detector. The correlation between true π^0 energy and reconstructed EM like energy in a 10 GeV pion shower is shown in Fig. 5 from the studies performed by Vasiliy Morgunov. More details on these studies can be found in: http://www.desy.de/∼morgunov/talks_articles/Full_Simulations.pdf.

5.0.4 `reassignment_small_clusters`

All hits left unassigned to the type under consideration after the previous steps are reassigned to the next type considered. The logic chain of the DeepAnalysis code is: find EM like clusters, EM hits not assigned are reassigned as TRK like hits. Find TRK clusters, TRK hits not assigned are reassigned as HAD like hits. Find HAD clusters, HAD hits not assigned in all the previous steps are tagged NEU like hits. NEU hits remain isolated, no NEU cluster exists. This last step of NEU assigning is performed in the routine `finalize` which has to be called at the end of the 3D clustering.

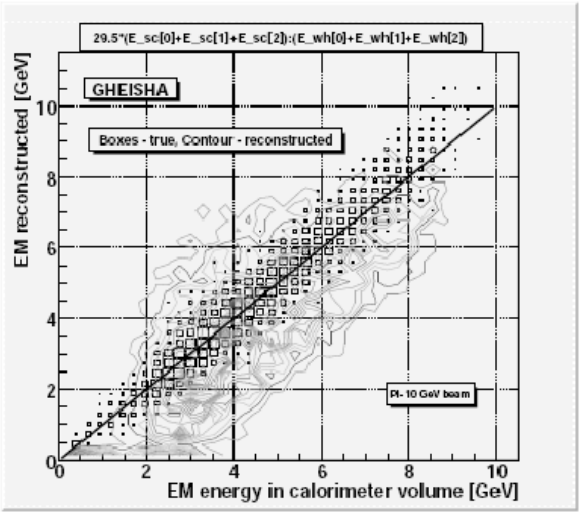

Figure 5: *Result of the tuning of DeepAnalysis parameters for π^0 energy reconstruction in a hadronic shower.*

5.0.5 `finalize`

Finalize acts on all collections of clusters identified in the 3D reconstruction. It uses the cluster eccentricity cut given by the parameter `ecc_trk_hadr_sel` to change TRK like to HAD like clusters if the number of hits in the cluster is larger than 2 and the eccentricity smaller than the cut. Clusters with number of hits less then 3 are destroyed and reassigned as NEU like hits.

After this last step the reconstruction is completed and the cluster classes can be retrieved for analysis.