Clustering pixel hits. An algorithm and some software.

Gianluca Aglieri Rinella
University of Palermo and Doctoral Student at CERN (LHCb)

September 5, 2005

1 Pixel hits, adjacency and clusters

Two pixel hits h_i, h_j of a set are adjacent if they share a side. Optionally two pixel hits sharing a vertex (corner) can also be defined adjacent.

Two pixel hits h_i, h_j of a set are connected if they are the same hit, if they are adjacent or if a set of hits $\{h_i, h_l, h_m, ..., h_j\}$ can be found such that h_i is adjacent to h_l, h_l to h_m and so on to h_j. A connected set is a set of connected hits.

Given a set of hits, a pixel cluster is a subset of connected pixels such that no other pixels of the set are connected to those in the cluster. The clustering problem is the problem of determining the clusters in a given set of hits.

The connection relation $\{(h_i, h_j)|\ h_i \text{ is connected to } h_j\}$ is an equivalence relation on the set of hits, being trivially reflexive, symmetric and transitive. The clusters of hits are the equivalence classes of the equivalence relation. The determination of the clusters coincides with the determination of the equivalence classes of the connection relation.

2 A clustering algorithm

An algorithm to extract the clusters from a set of pixel hits $\{h_1, h_2, ..., h_N\}$ is described in the following. Refer to computer science literature for better algorithms (**Union-Find**).

The algorithm is completed in two phases: the first phase build sets of connected hits. The second phase merges the connected sets found in the first phase if they contain couples of adjacent hits. At the end of the second phase the list of clusters (equivalence classes) is obtained.

- First phase: given the N *distinct* hits $\{h_1, h_2, ..., h_\mathrm{N}\}$ build sets of connected hits:

1. Cycle over all hits h_i $i = 1...\mathrm{N}$
2. For each h_i scan over the list of connected sets determined till now. Check if h_i is adjacent to any of the hits in the k-th connected set; if true, add the hit to that set.
3. If the hit was not added to any of the connected sets found till this point, then add a new connected set at the end of the list, with the i-th hit as only hit.

At the end of the first phase a list of connected sets is determined. They are not the clusters because there can be two different connected sets containing connected hits, so being subsets of the same cluster. There is the need to scan the list of connected sets found till this point and merge those sets satisfying this condition into larger connected sets.

- Second phase: Let C_k $k = 1...\mathrm{M}$ be the M connected sets determined after first phase. Observe that some of them may be single hit sets.

  1. Cycle over k backward from last to second: $k = M...2$
  2. For each k, cycle over j forward from first to *(k-1)*-th: $j = 1...(k-1)$. If C_k and C_j have a couple of connected hits, then merge the C_k with C_j, appending the hits of C_k to the latter; delete the kth connected set C_k from the list. Else do nothing.

     Notice that the order of the scanning is such that even if C_k was merged with C_j, the updated C_j will be later compared with connected sets with index smaller than j.

  At the end of the second phase, all possible connected sets have been merged and the list of connected sets is now the list of clusters.

3 Software

Software has been written to support the analysis of HPD data. Three C++ classes were defined and implemented:

1. My_hit:

   It is a class modeling a simple hit, i.e. wrapping the row and col coordinates. It provides a set of boolean member functions to check if a hit is adjacent to another. Example: *first_hit.IsAdiacentTo(second_hit)* returns true if the adjacency condition is verified. Code is to be modified in this function if pixel hits sharing a vertex have to be considered adjacent.

2. My_whit:

   It is a class inheriting from the previous (i.e. provides same data members and interface) adding storage for a weight attributed to the pixel hit. It is useful to store information on pixel hit more than once. It can be used practically always, given the constructor attributes an unitarian weight to the hit if not specified differently.

3. My_cluster:

   It is a class built around a simple data member, a vector of My_whit objects. It provides a member function to add hits to the cluster that check for adjacency with at least one of the hits already in the cluster. Forced insertion is also supported. Straightforward extraction of statistics on a cluster (total weight, centre of mass, average row or column coordinate, root mean square deviation) is also provided.

4. Other functions in auxiliary namespaces:

   void My_hit_tools::Eliminate_twins(vector<My_hit > &hit_list) eliminates from the vector of hits redundant entries, i.e. hits with same coordinates. This should be a pathologic condition.

   void My_whit_tools::Eliminate_twins(vector<My_whit > &whit_list) eliminates from the vector of weighted hits redundant entries, i.e. hits with same coordinates. This should be a pathologic condition. The weights of eventual multiple instances are added.

   vector<My_cluster > clustering_tools::Clustering (vector<My_whit > hits_list) gets in input a vector of weighted hits and realizes the clustering with the algorithm previously described, returning the vector of clusters.

   vector<My_cluster > clustering_tools::Clustering (vector<My_hit > hits_list) overloading the previous method to treat a vector of My_hit.

To use all of the previous, it is enough to add the line #include ".//My_cluster.C" in the analysis program. There was not the time nor the need to provide separation of headers from implementation; apologies.