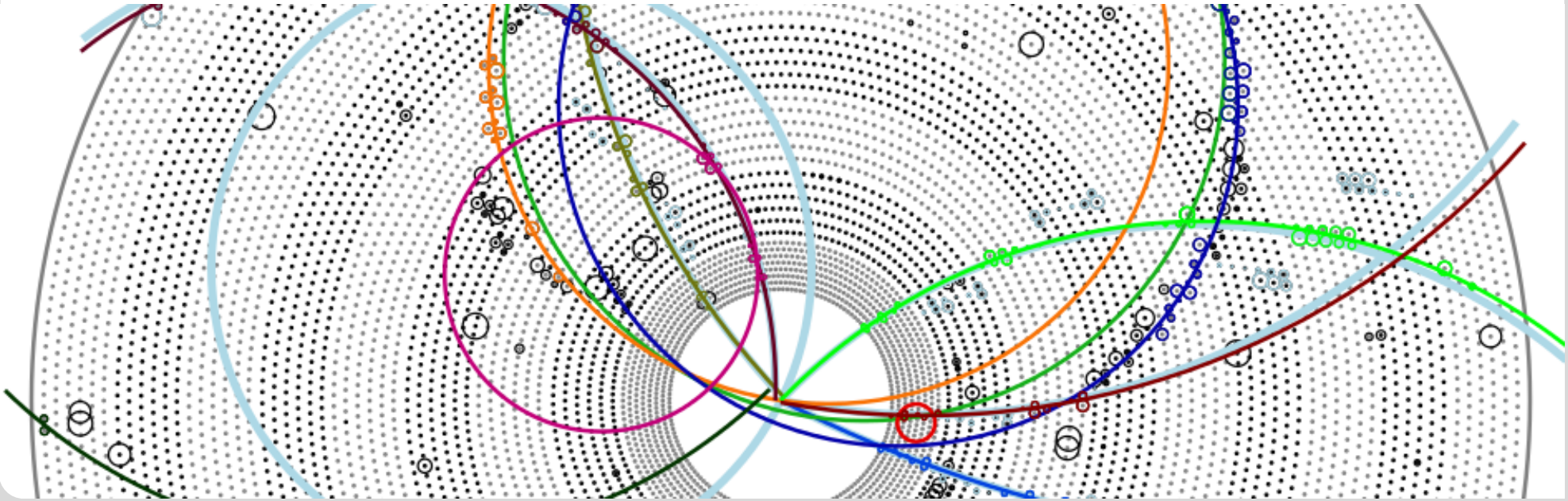
Karlsruher Institut für Technologie

CDC Finder Preparation and Analysis

F2F Tracking Vienna

Nils Braun | 21.04.2015



Postprocessing after the Legendre Track Finder

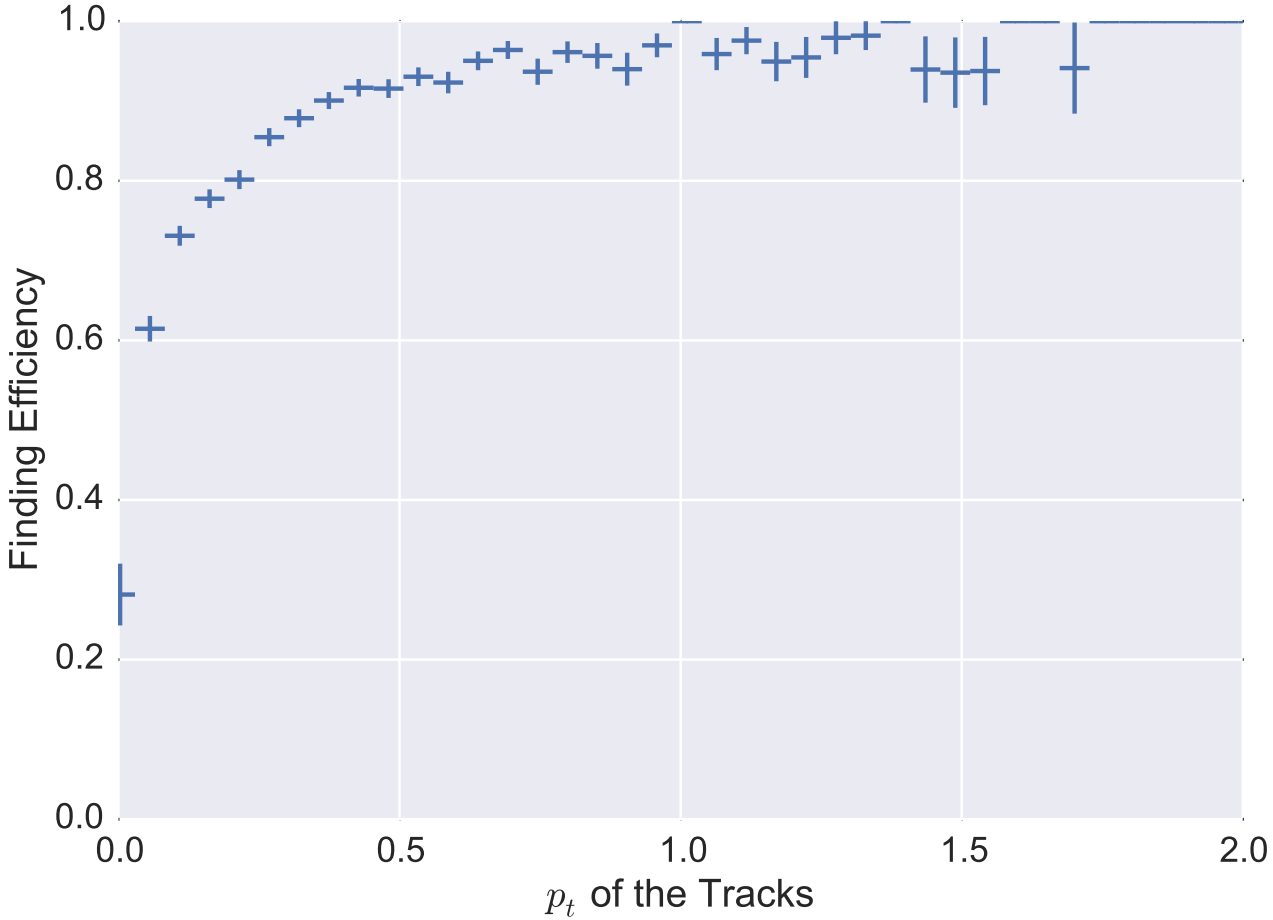
“Simple” algorithms implemented and refactored:

- Deletion of hits from a track with calculating a certain “index” (as a function of the distance between track and hit). Something like χ^2.
- Reassignment of hits between already found tracks.
- Merging of tracks with a simple circle fit. Cleanup of “bad” hits.

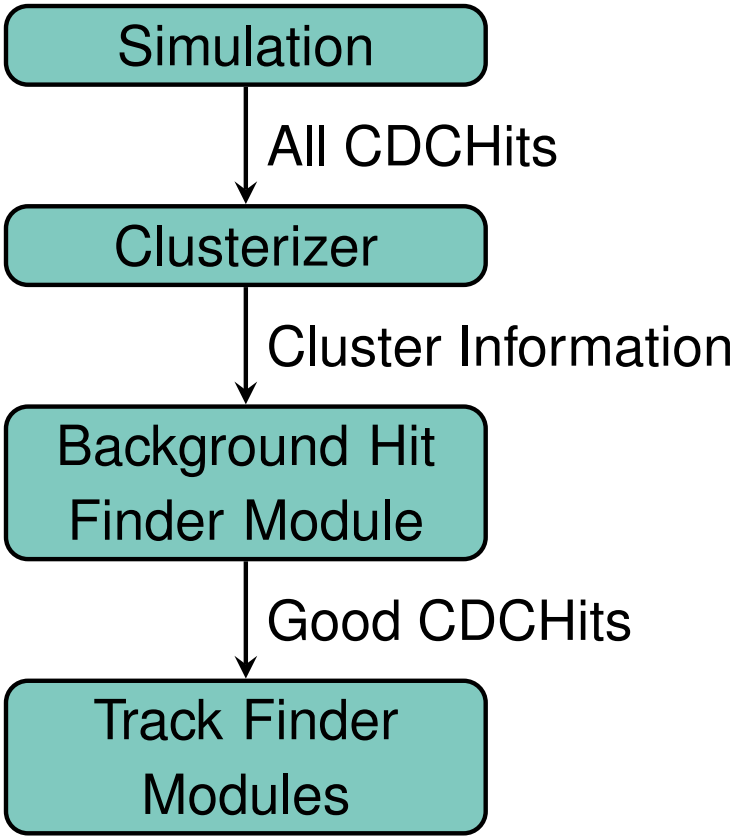
Improvements on fake- and clone-rate:

	Without any postprocessing	With postprocessing	Stereo Histogramming
Fake-Rate	29.22 %	23.89 %	26.04 %
Clone-Rate	23.25 %	11.51 %	10.55 %
Efficiency	86.02 %	85.34 %	83.93 %
Hit Efficiency	48.87 %	53.49 %	79.43 %

Results


1.0
0.8
0.6
0.4
0.2
0.0
Finding Efficiency
0.0
0.5
1.0
1.5
2.0
p_t of the Tracks

Background Hit Finder - Idea



- With the realistic TDC calculation the decision if a hit is background or not gets harder.
- Proposed solution:
  - Clusterize the hits with the clusterizer from the LocalTrackFinder.
  - Decide with a trained BDT whether a cluster is background or not.
  - Use only good hits as input for all following track finder (e.g. the legendre track finder).

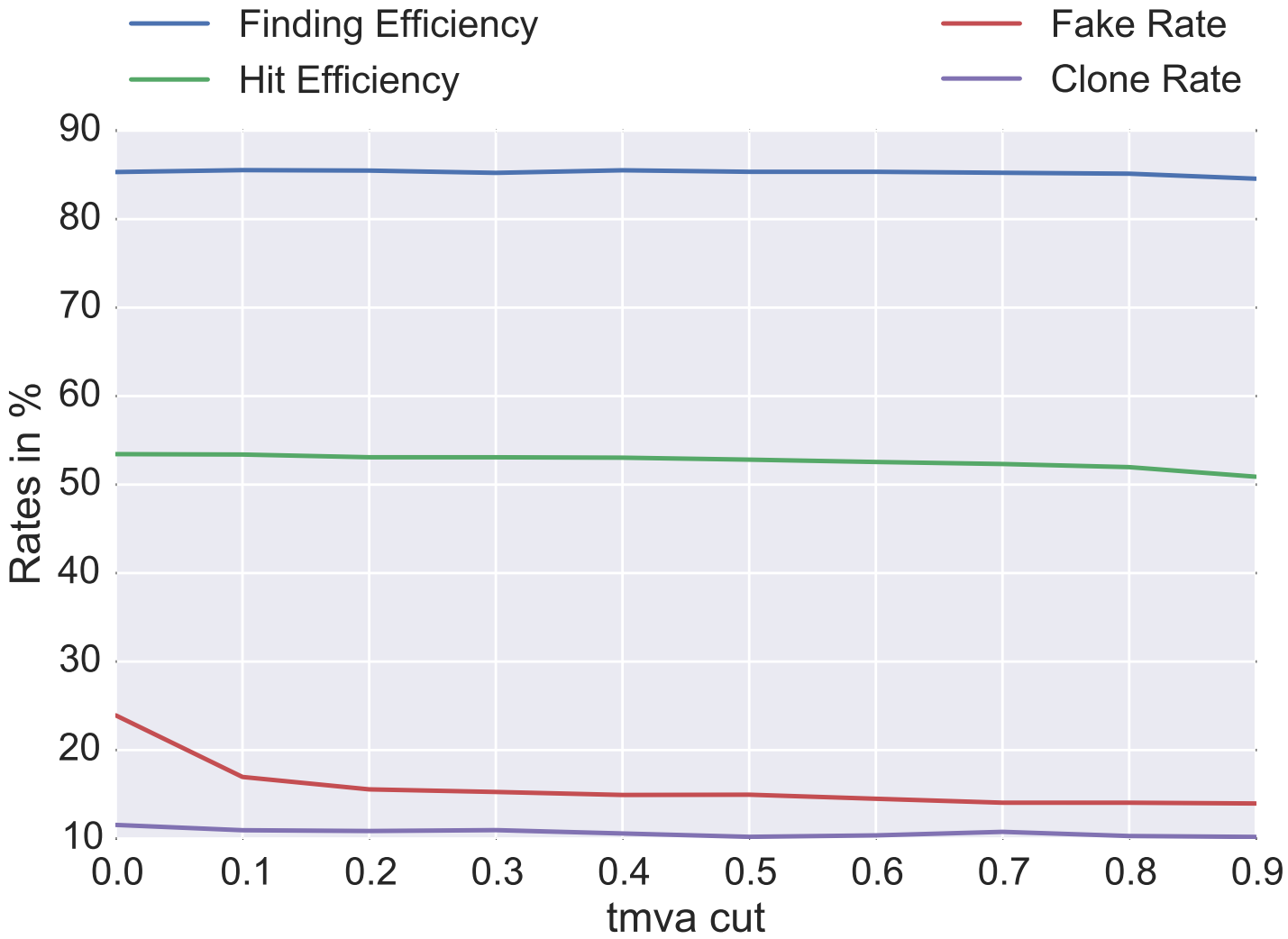
The BackgroundHitFinderModule is - thanks to Oliver - now part of the local track finder!

Background Hit Finder - TMVA

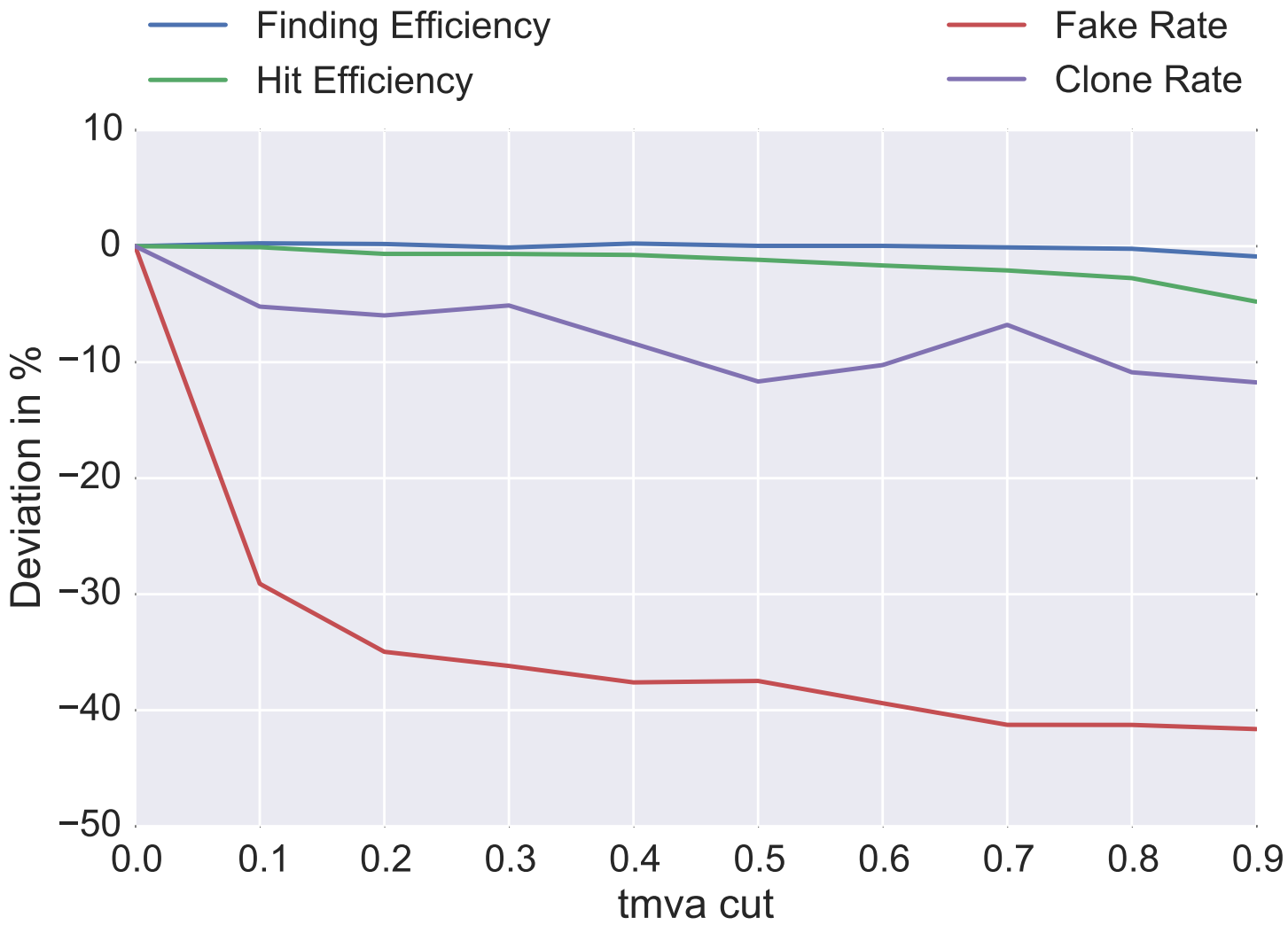
The BDT uses the following input variables:

- `superlayer_id` is unique because a cluster is in one superlayer
- `is_stereo` of the superlayer
- `size` = number of hits in the cluster
- `total_n_neighbors` of hits
- `avg_n_neignbors` of hits
- `total_drift_length` of the hits
- `total_inner_distance` between the IP and the hits
- `variance_drift_length` of the hits
- `distance_to_superlayer_center` = superlayer center - mean of the positions of the hits
- `mean_drift_length`
- `mean_inner_distance`

Background Hit Finder - Results



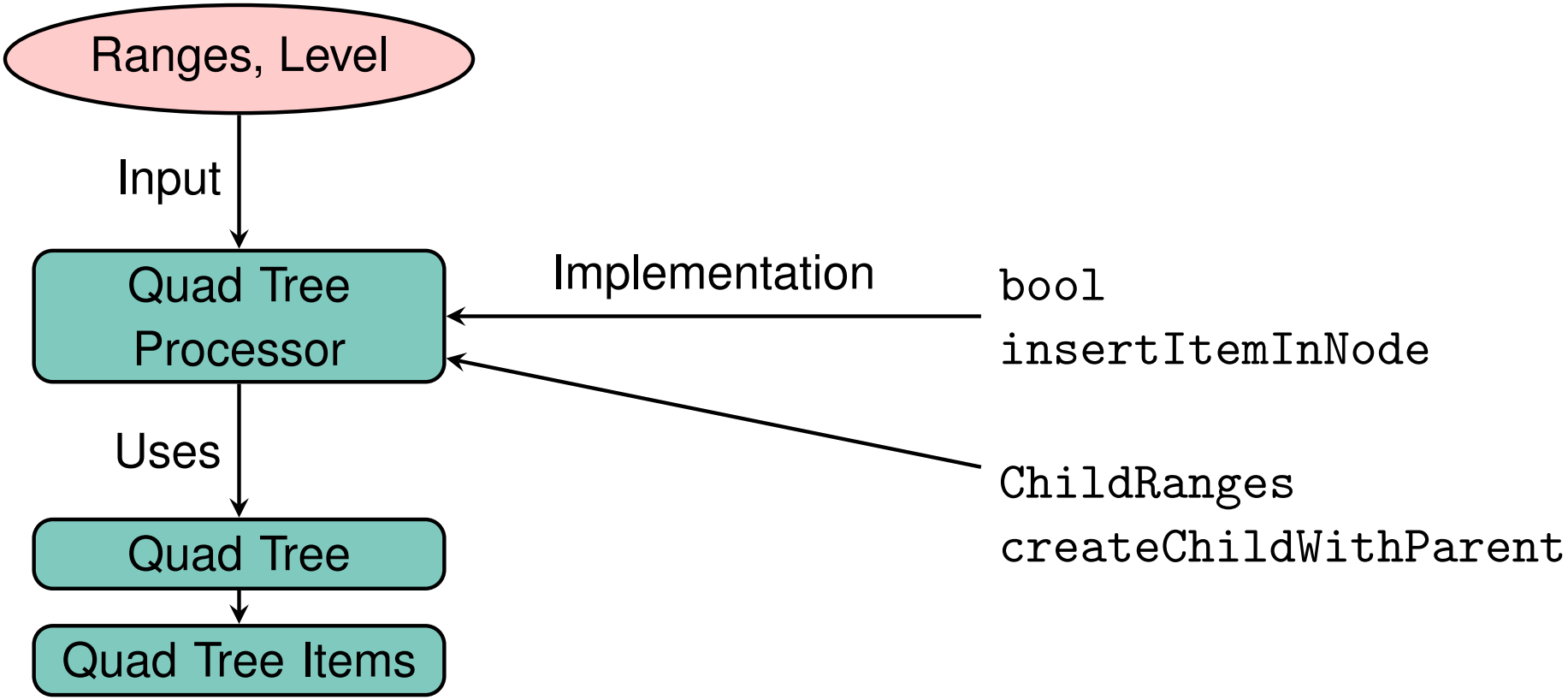
Background Hit Finder - Results



Quad Tree Processor Template

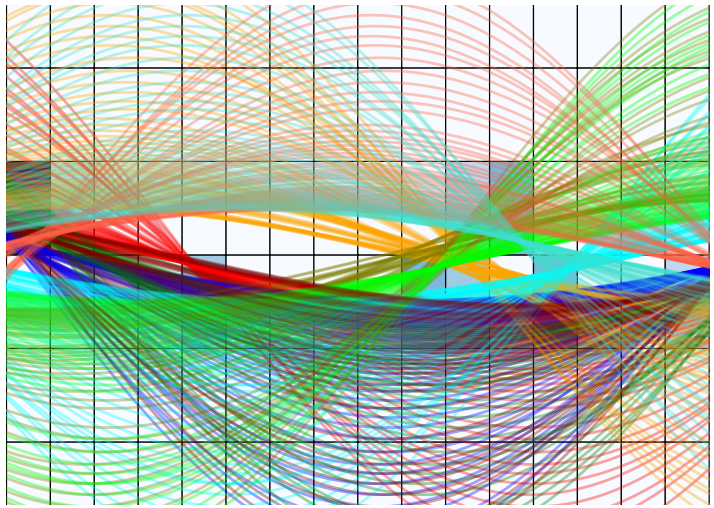
- Some time ago Viktor and Thomas have implemented a templated version of the QuadTree itself.
- But: without writing an own filling procedure you can not put in your own item class.
- I have implemented a templated abstract QuadTreeProcessor and a general templated QuadTreeItem class.
- As a first test I could write a SegmentQuadTreeModule in roughly 100 lines of code.

Quad Tree Processor Setup

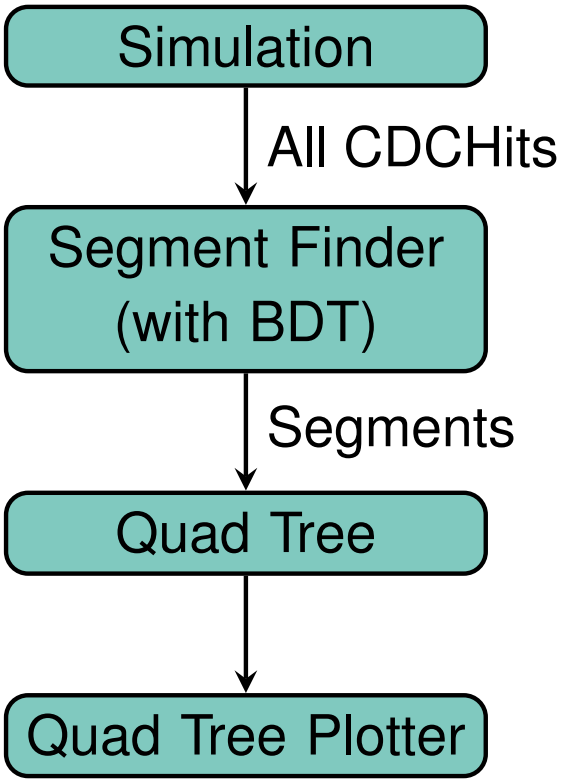


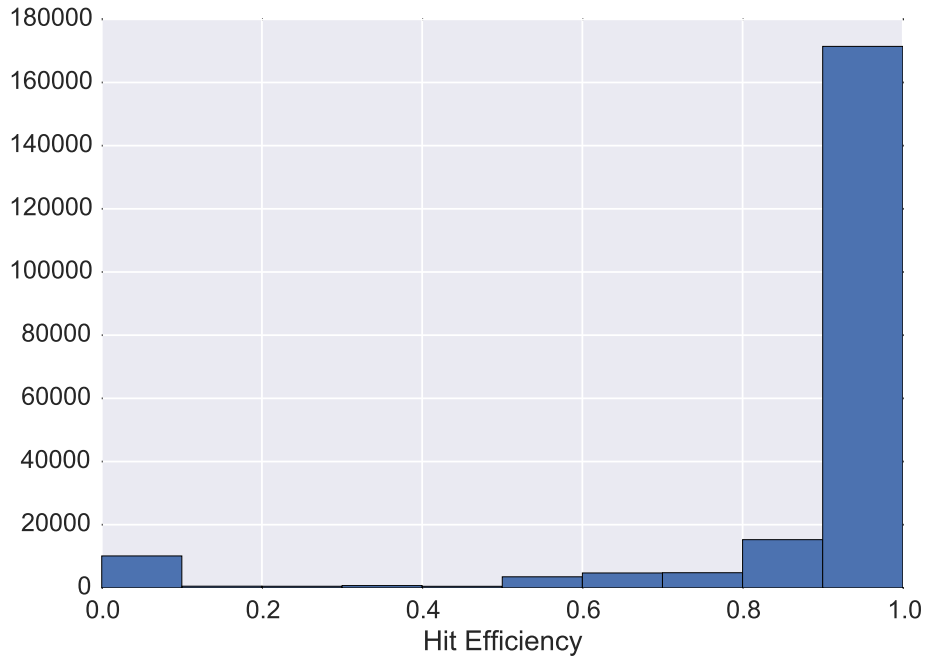
Quad Tree Processor Plotter

- With the new QuadTreeProcessor you can also define a “hook” to get to the information of the items when filling.
- This allows some debug output and afterwards some plotting.
- Be aware that this has a huge impact on performance!
- The plotting is done by a python script and `matplotlib`.



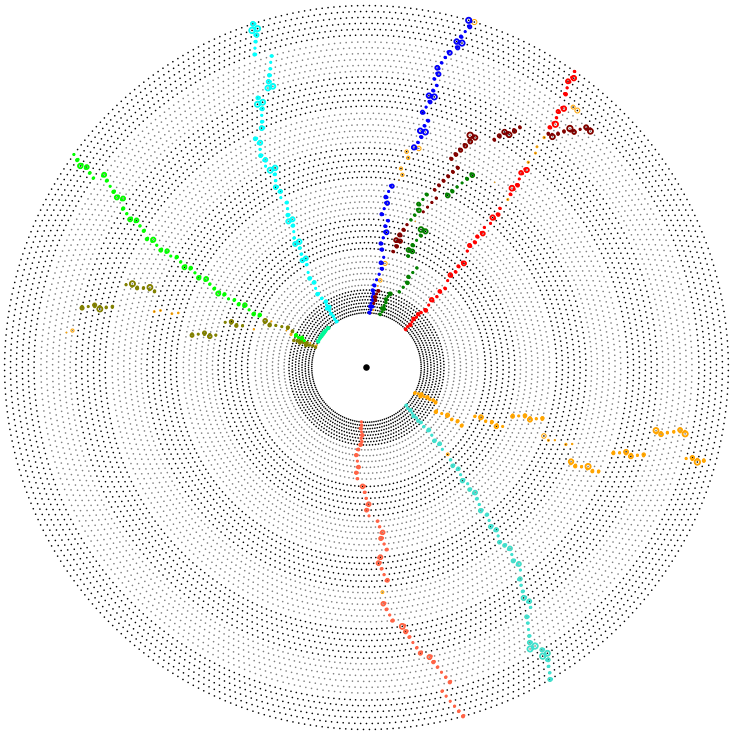
Motivation and Workflow

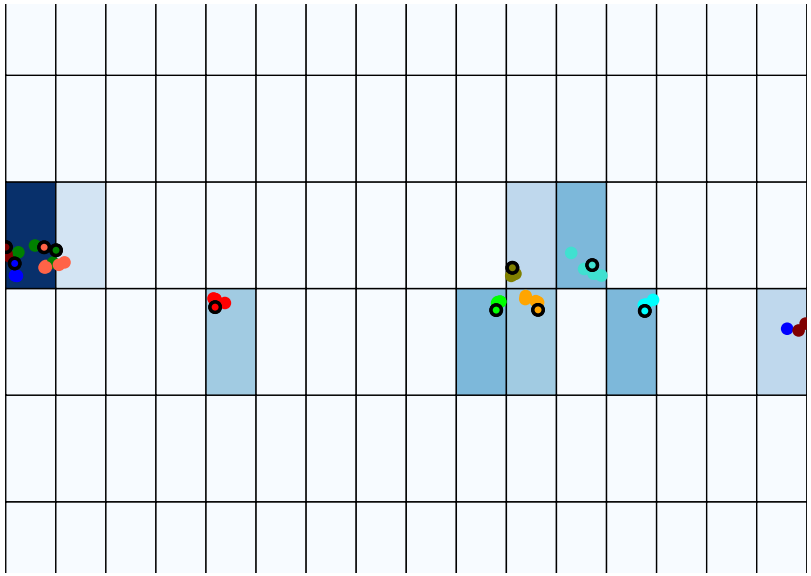




The Segment Finder creates Segments with a very high purity.

Preliminary Results





Preliminary Results