Pattern Recognition using Hough Algorithm

IEP SAS, Košice
TUKE, Košice

Jozef Vasilko
Michal Vrábel
Marian Putis
Pavol Bobik

Hough transform for lines

- detects lines in image
- parameter space representation
  - slope-intercept form of line equation
    - $n = y - mx$
    - (n, m) parameter space
    - m can be huge for near vertical lines
    - no representation for a vertical line
  - polar form of line equation
    - $r = x * \cos\theta + y * \sin\theta$
    - (r, Θ) parameter space
  - line in a image corresponds to the point in parameter space

Algorithm (2D)

- input image
- create parameter space
  - create two arrays rArray[R], θArray[F] containing discretized parameter space r, Θ using sampling steps dr, dΘ
  - every line should have unique point in parameter space , therefore: $\theta \in [0, \pi) \wedge r \in R$, or $\theta \in [0, 2\pi) \wedge r \geq 0$ etc.
- create counter array A(R, F)
- if input isn't binary, use threshold to do so
- for each pixel P(x, y) with value == 1 (binary input) or >threshold do
  - $\text{rValue} = x * \cos(\theta\text{Array}[i]) + y * \sin(\theta\text{Array}[i]), i \in 0, \ldots, F - 1$
  - find index k of rArray so that rArray[k] is closest to rValue
  - increment A(k, i)
- find maxima in A(R, F), decide the existence of lines and make output (you can get pixels by computation or from memory, in case they were stored before)

Algorithm (3D)

- problem of finding line in 3D space broken into two problems of finding line in 2D space
  - input --> runHough(x, z) -->runHough(y, z)

Our algorithms for pattern recognition

- Combination of 2D and 3D Hough transform for lines algorithms
- First algorithm searches shower with 2D Hough, using XY space, in data above with signal value above the given threshold and with 3D Hough, using XYTime space, it selects the data from 2D Hough output with wanted time characteristic
- Second algorithm uses 2D Hough just for selecting the minimal area, that contains a shower, and using 3D Hough it searches for data of shower

First algorithm (1)

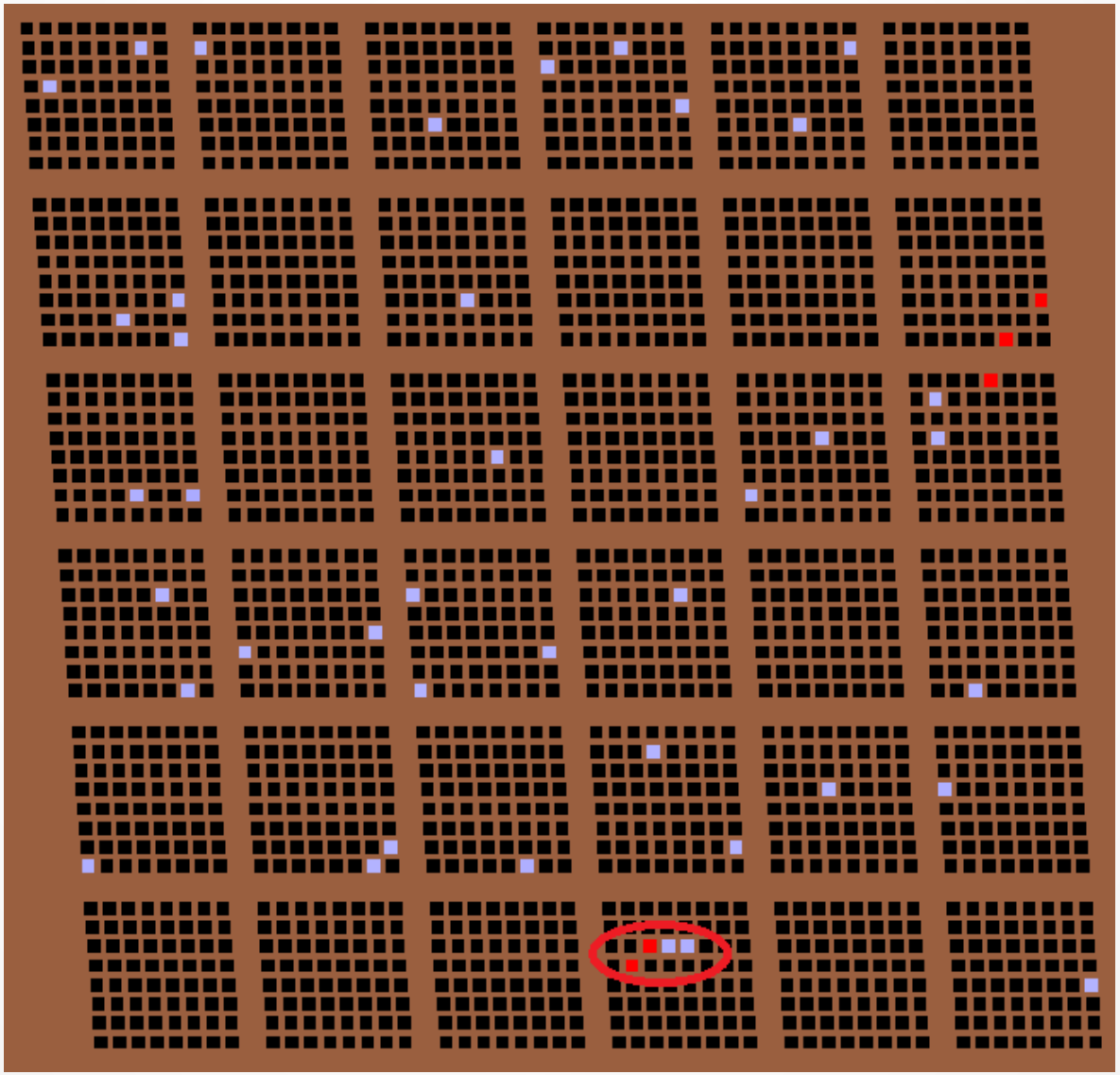
- input data
- select data with signal above the given threshold
- select unique pixels (x, y coordinates)
- run 2D Hough in XY space
- run 3D Hough in XYTime space (its input is output from 2D Hough)

First algorithm (2)

- Disadvantages
  - could be good only for finding incline angle
  - the more pixels displayed the less possibility that the line will be going along the shower
- Improvements
  - weighted points
  - finding the core pixels and giving them enough weight

Example

weight = max signal that occurs on given position

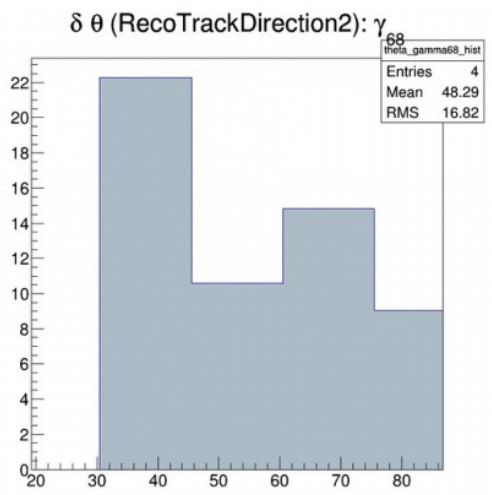
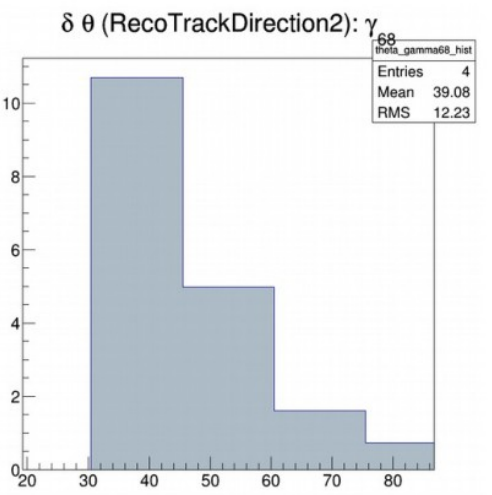
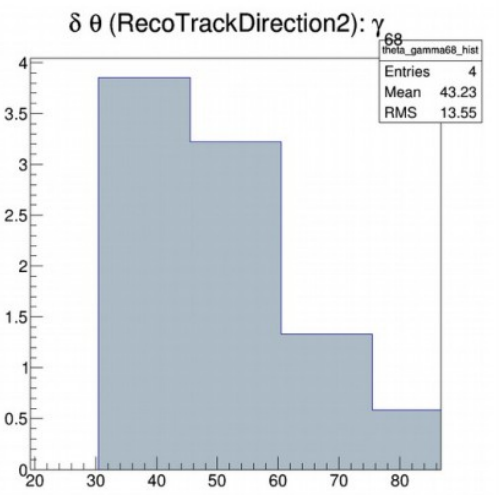
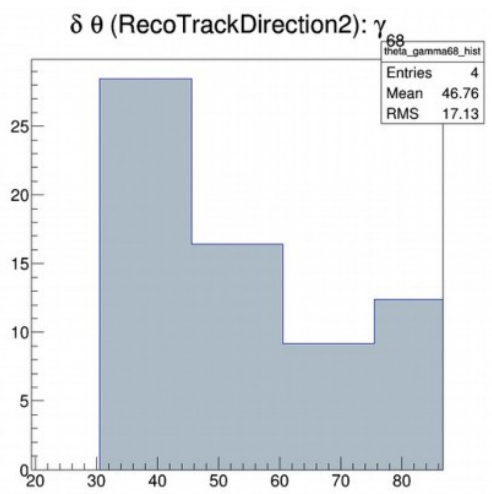
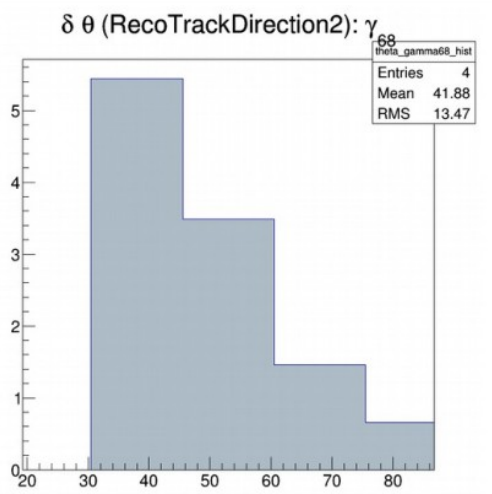
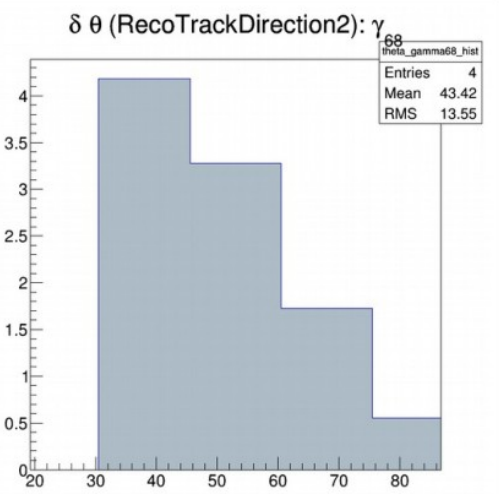


γ_{68} for $\theta = 45°$ (1e20 eV)

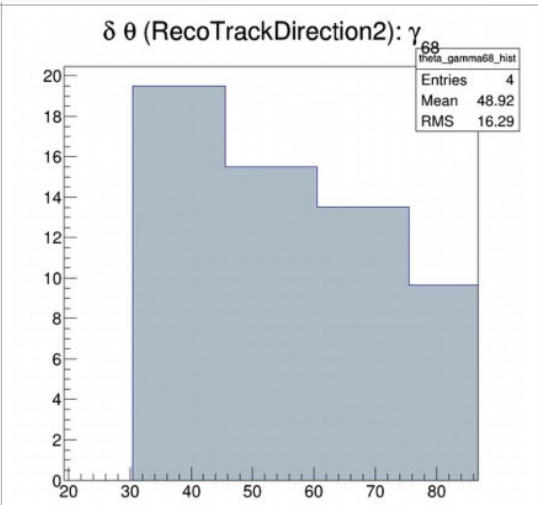
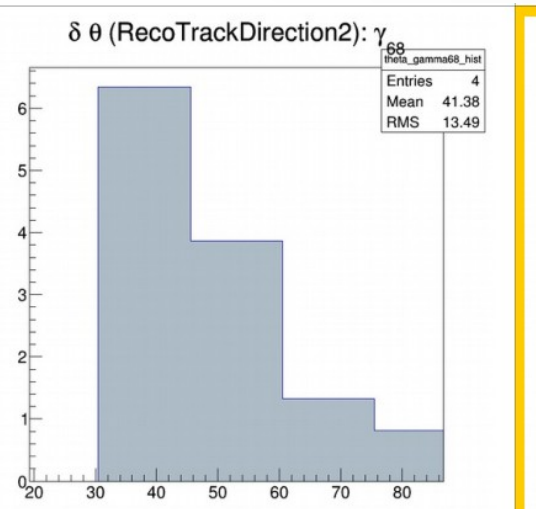
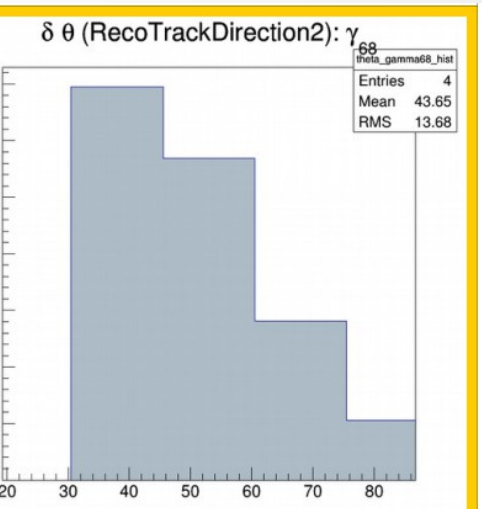
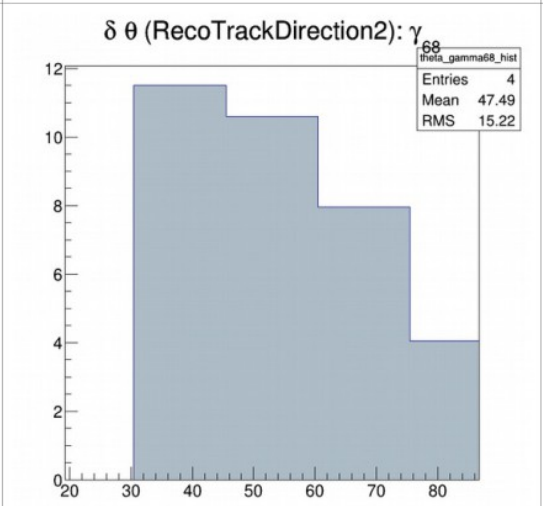
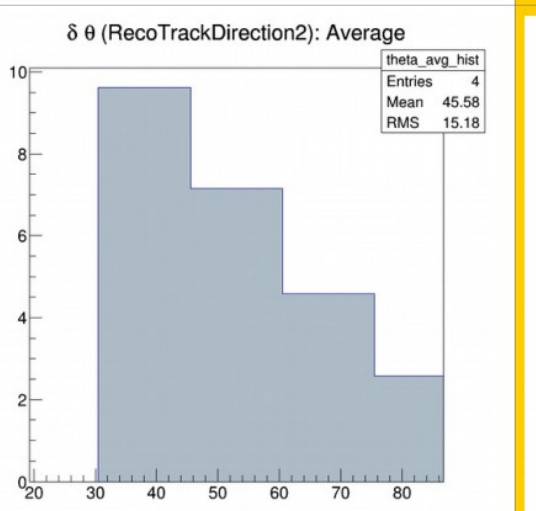
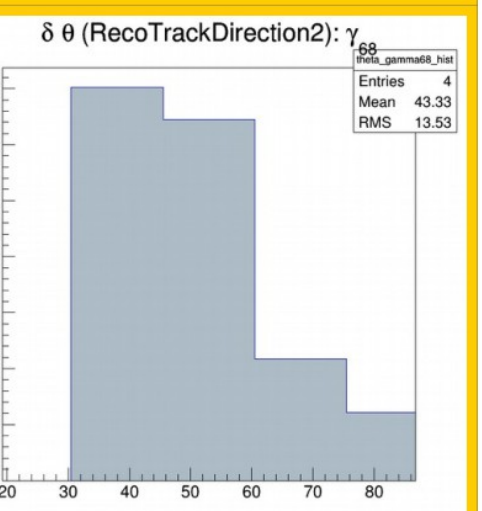
γ_{68}	threshold = 4	threshold = 5	threshold = 6
dr = 1 mm	10.59	4.97	3.22
dr = 1.5 mm	16.39	3.49	3.28
dr = 1.98 mm	15.51	3.87	**2.84**
dr = 3 mm	10.59	4.97	3.22

Simulated impact: xMin=-270km xMax=270px yMin=-200km yMax=200

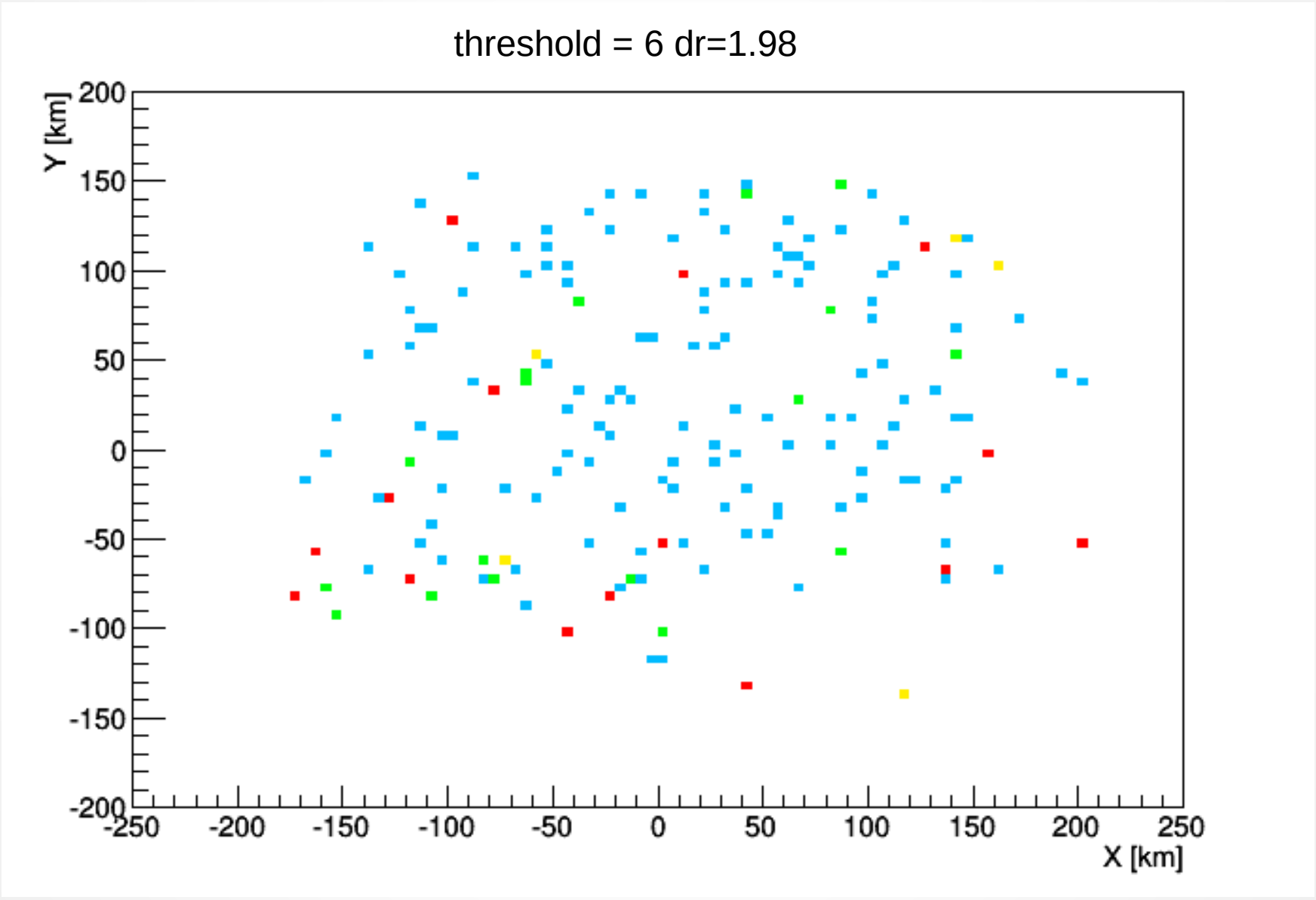
Multiple *dr* and *threshold* combinations

δ_{68}	threshold = 4	threshold = 5	threshold = 6
dr = 1 mm			
dr = 1.5 mm			

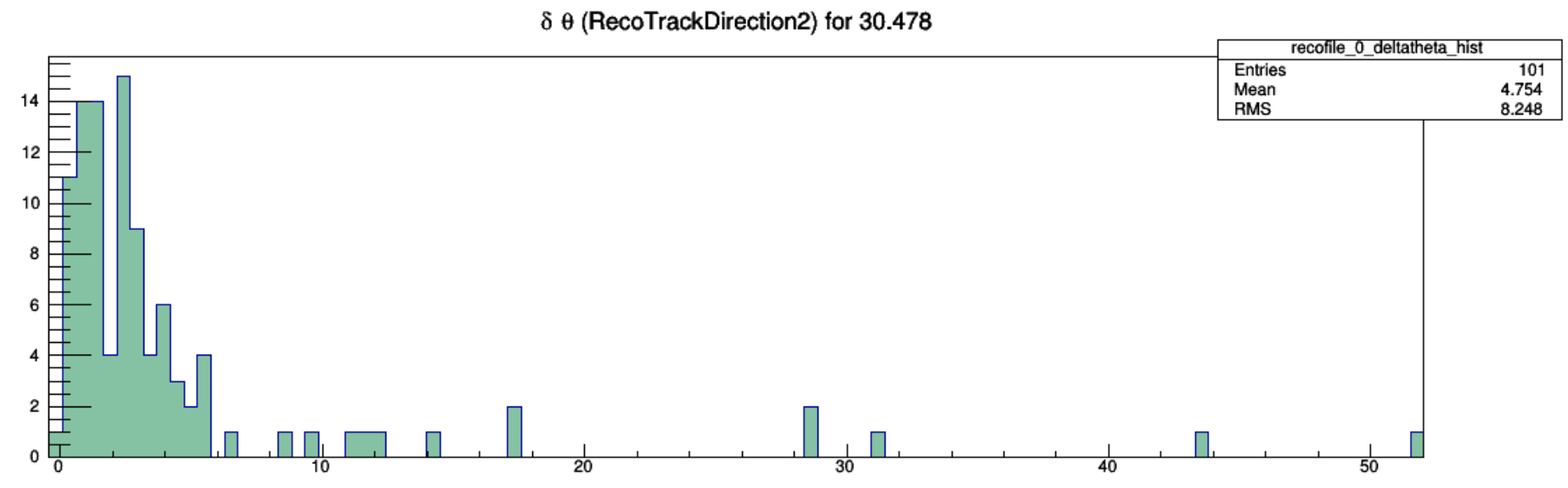
Multiple *dr* and *threshold* combinations

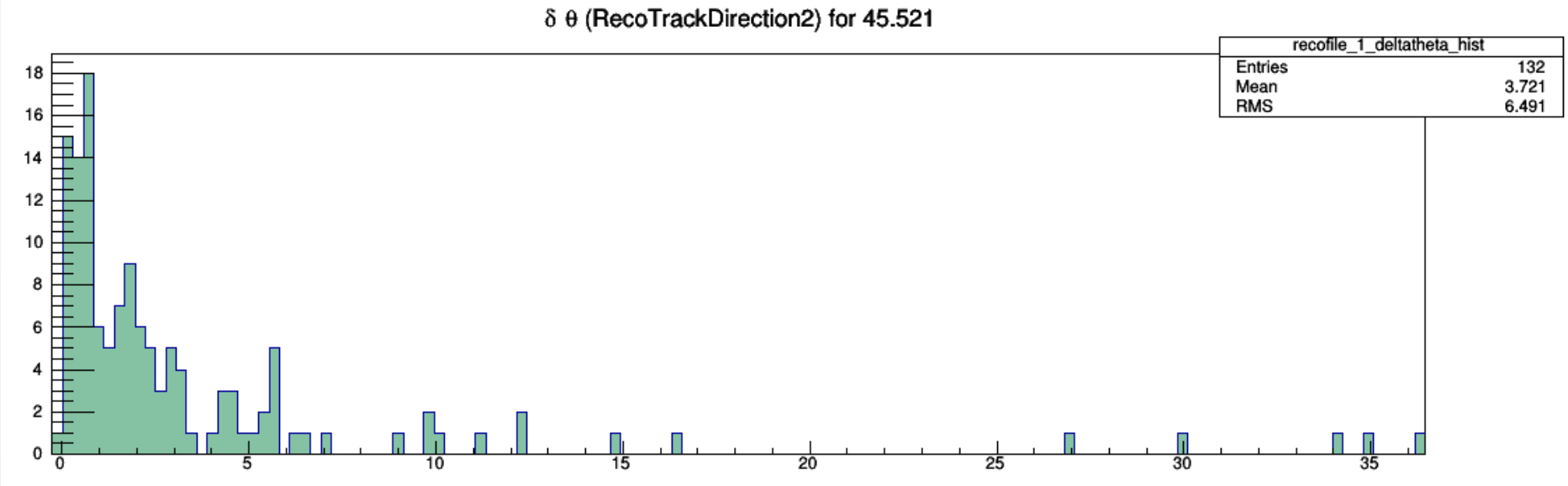
δ_{68}	threshold = 4	threshold = 5	threshold = 6
dr = 1.98 mm			
dr = 3 mm			

Distribution of γ vs X_{max} in full JEM-EUSO FoV

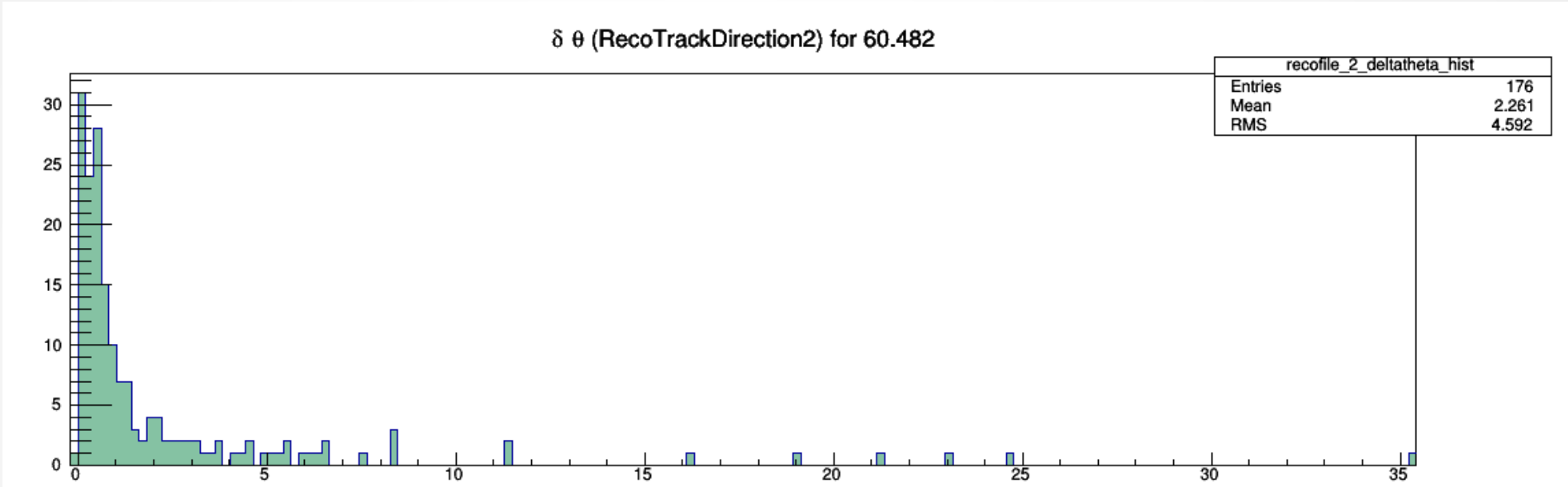


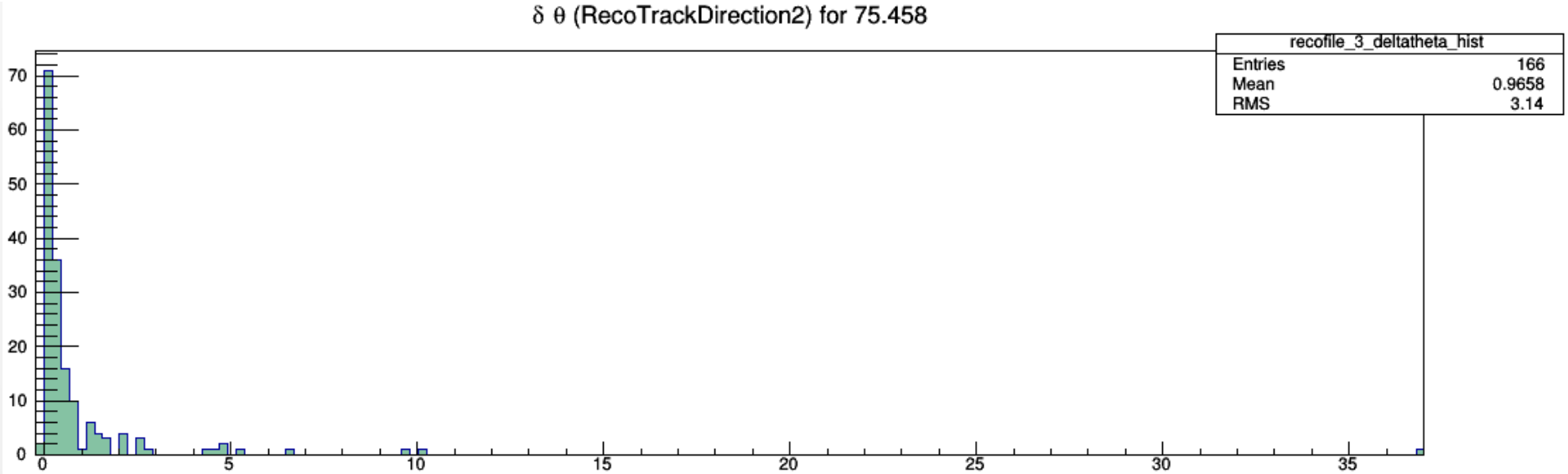
δθ for threshold=6 dr=1.98 (1)





δθ for threshold=6 dr=1.98 (2)





Numbers of events accepted by TrackDirection2

(RecoTrackDirection2::GetQuality() != -1)

θ = 30°

γ_{68}	th4	th5	th6
1 mm	**111**/337	**122**/337	**93**/337
1.5 mm	**127**/337	**128**/337	**103**/337
1.98 mm	**137**/337	**129**/337	**101**/337
3 mm	**129**/337	**146**/337	**113**/337

θ = 45°

γ_{68}	th4	th5	th6
1 mm	**160**/335	**203**/335	**148**/335
1.5 mm	**140**/335	**169**/335	**136**/335
1.98 mm	**138**/335	**187**/335	**132**/335
3 mm	**160**/335	**203**/335	**148**/335

θ = 60°

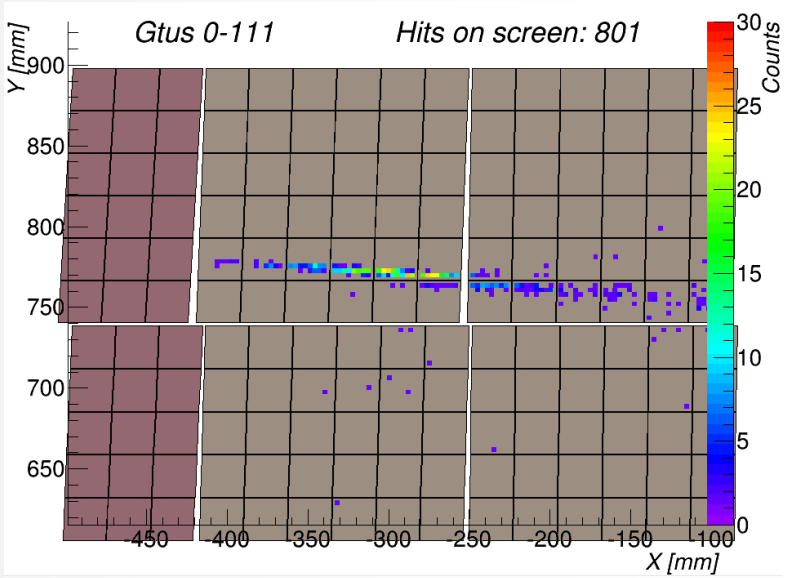
γ_{68}	th4	th5	th6
1 mm	**165**/337	**198**/337	**167**/337
1.5 mm	**182**/337	**209**/337	**176**/337
1.98 mm	**174**/337	**211**/337	**176**/337
3 mm	**191**/337	**234**/337	**196**/337

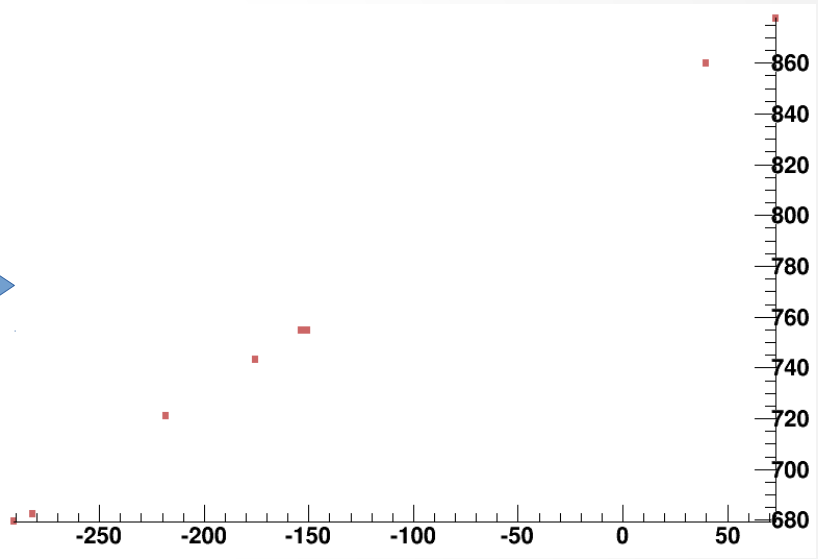
θ = 75°

γ_{68}	th4	th5	th6
1 mm	**129**/217	**157**/217	**158**/217
1.5 mm	**145**/217	**165**/217	**165**/217
1.98 mm	**125**/217	**170**/217	**166**/217
3 mm	**146**/217	**169**/217	**168**/217

Examples (1)
(threshold 6 dr 1.98)

Failed reconstruction



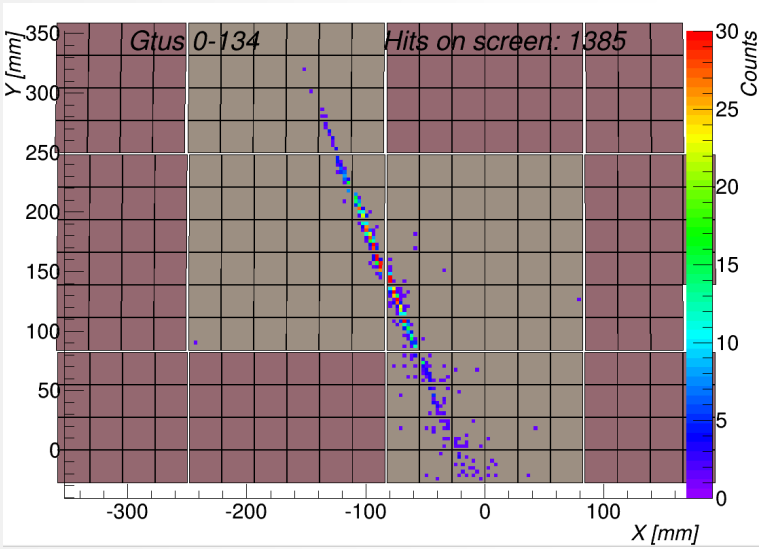


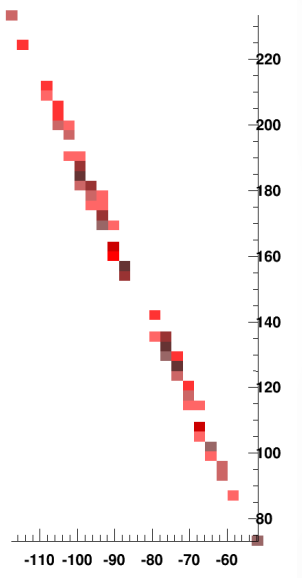
Energy: 1e20 eV
Theta: **75.92°**
Phi: 357.11°

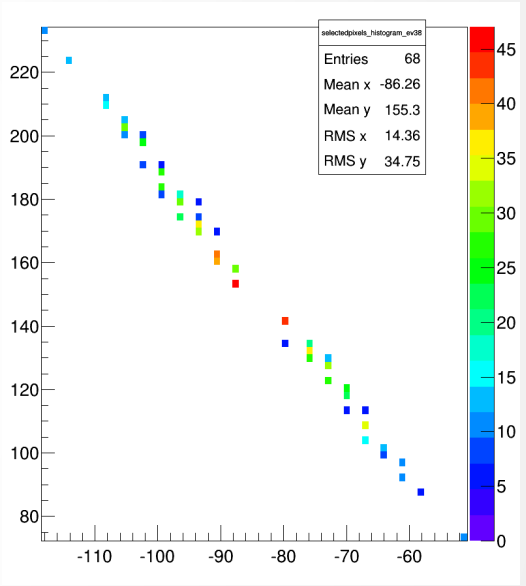
RecoTrackDirection2::GetQuality() == -1
No angle was reconstructed
by module RectoTrackDirection2

Examples (2)
(threshold 6 dr 1.98)

Successful reconstruction





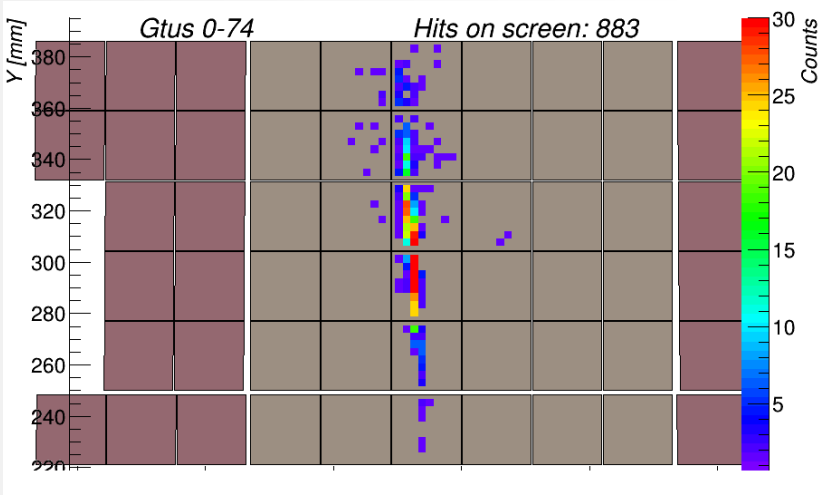


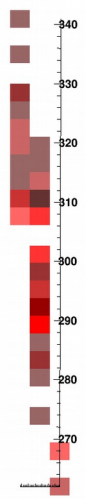
Energy: 1e20 eV
Theta: 75.53°
Phi: 291.7°

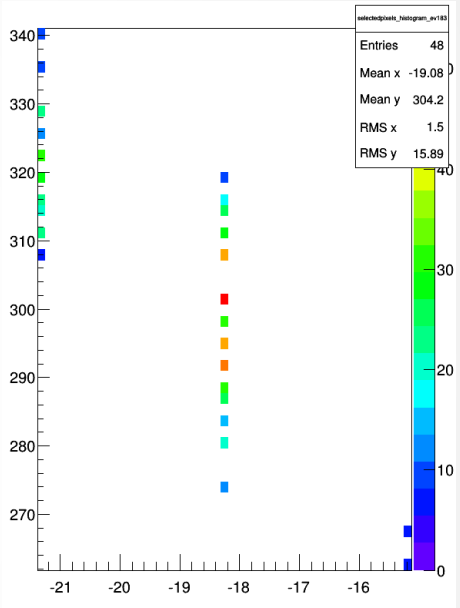
Reconstructed by module RectoTrackDirection2:
Theta: 75.6248
Phi: 291.608
ErrorTheta: 0.0936613
ErrorPhi: -0.150757

Examples (3)
(threshold 6 dr 1.98)

Successful reconstruction







Energy: 1e20 eV
Theta: 60.49°
Phi: 93.94°
X1: 35.457
Xmax: 880.457

Reconstructed by module RectoTrackDirection2:
Theta: 60.60°
Phi: 94.01°
ErrorTheta: 0.11
ErrorPhi: 0.07

Reconstructed by module RecoPmtToShower:
Energy: 1.34913e+20 eV
EnergyError: 7.22513e+18 eV
Xmax: 886.335
XmaxFit: 1046.49

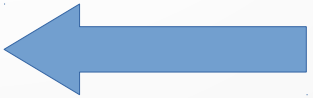
Result data in ESAF

```
void KeHoughModule::WriteSingleCluster() {
    if ( !fHittedFinal.size() ) return;

    // write the cluster
    EusoCluster *cl = new EusoCluster();
    for( UInt_t i=0; i<fHittedFinal.size(); i++ ) {
        cl->AddPoint( fHittedFinal[i] );
    }
    fClusters.push_back(cl);

    // add data to event
    vector<Int_t> *pix = &fHittedFinal;
    MyData()->Add("CluPixels", pix);
    vector<EusoCluster*> *pclu = &fClusters;
    MyData()->Add("Clusters", pclu);

    // add global data to the event
    RecoGlobalData* data = fEv->FindCreateGlobalData("PatternRecognition");
    data->CleanMaps();
    data->SetIssuer(GetName());
    data->Add("SelectedPixels",&fHittedFinal);
}
```

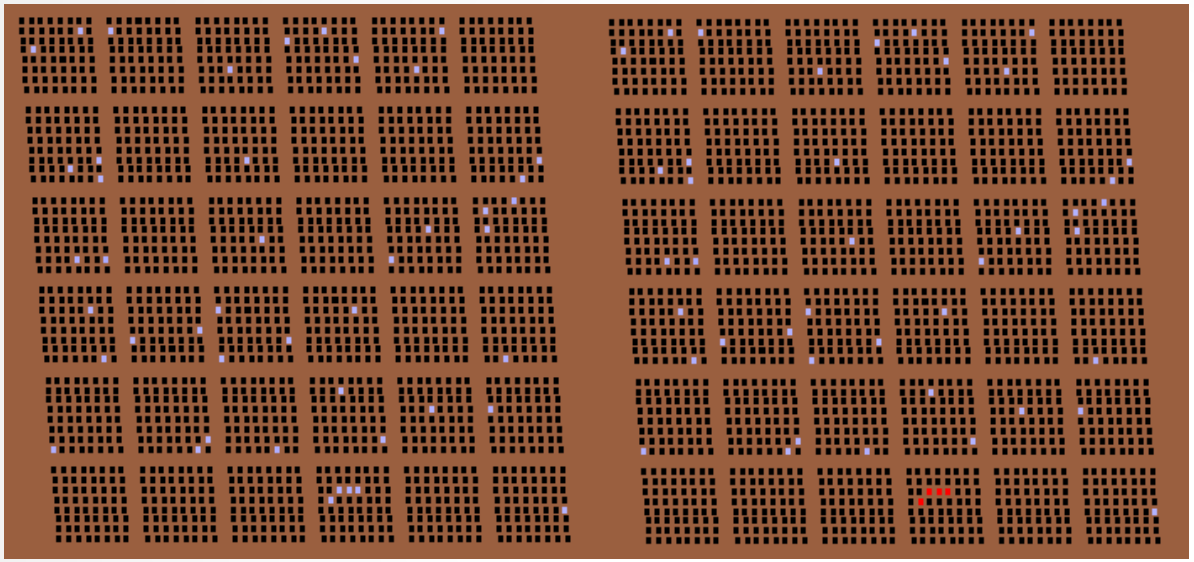


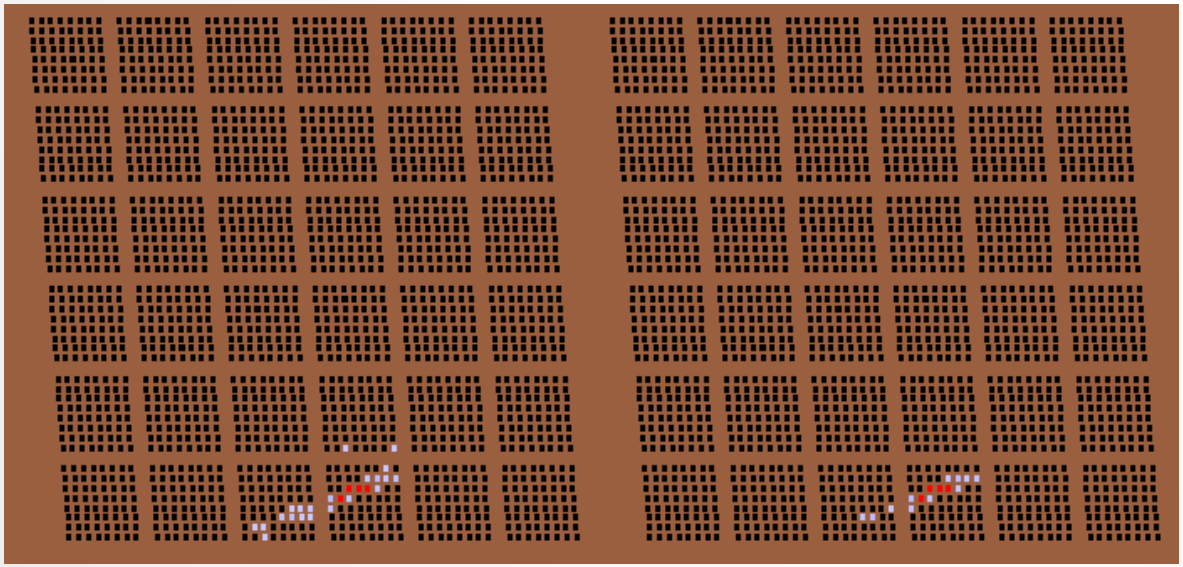
Read by
TrackDirection2Module,
PmtToShowerReco

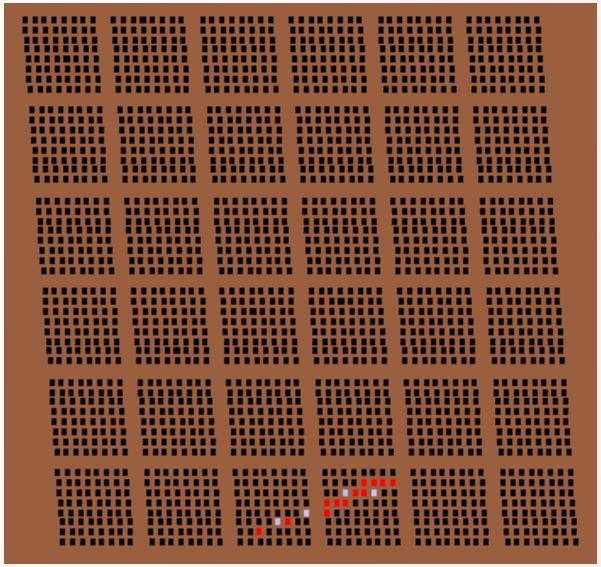
Second algorithm

- 2D and 3D Hough - count matrix is incremented by the value of signal of pixel data
- input data
- find the core of the shower or its part with the greatest sum of signal
  - set the threshold so that count of displayed pixels is as soon as possible at least 1%
  - find all groups of neighboring pixels and select the one with the greatest sum of signal
  - if it happens no group was selected (could be set minimal count of points for accepting group), decrement threshold
- add some constant value to the signal of the pixels of found group (in case they aren't "visible" enough)
- run 2D Hough to get line going along the core, then select the data that are within the rectangle border given by found line and some given width
- shift positions of pixels in positive quadrant
- set the threshold as before and run 3D Hough, you get the core of the shower and multiply signal value of pixels of that core by some big enough constant value (in next computation, it will give enough weight to the lines that contain these data)
- set the threshold and run 3D Hough in XYTime space

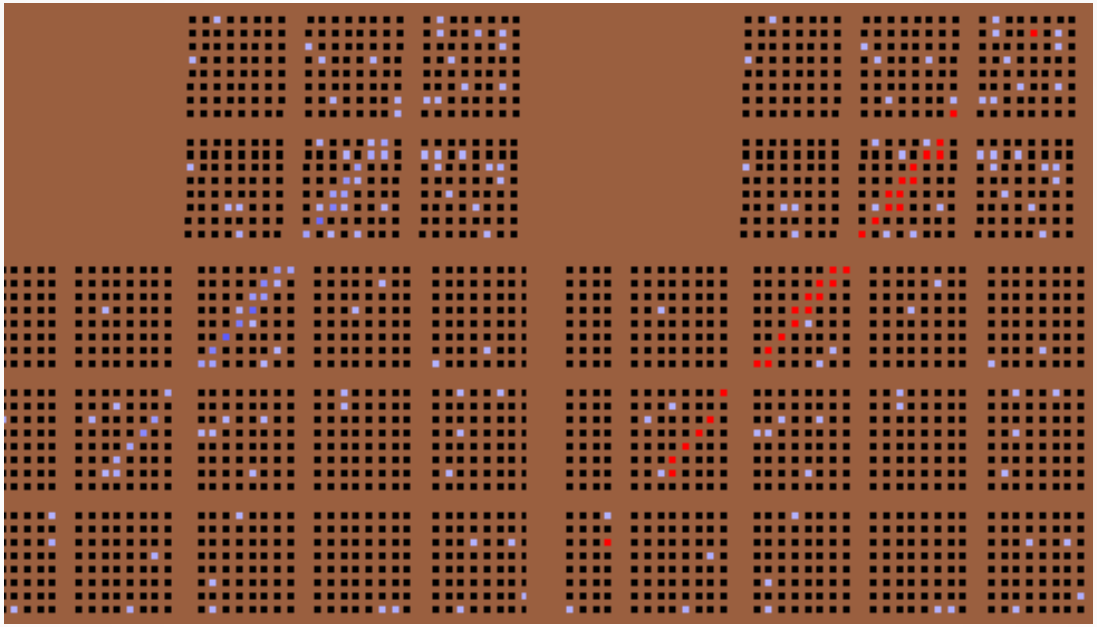
Example (1)

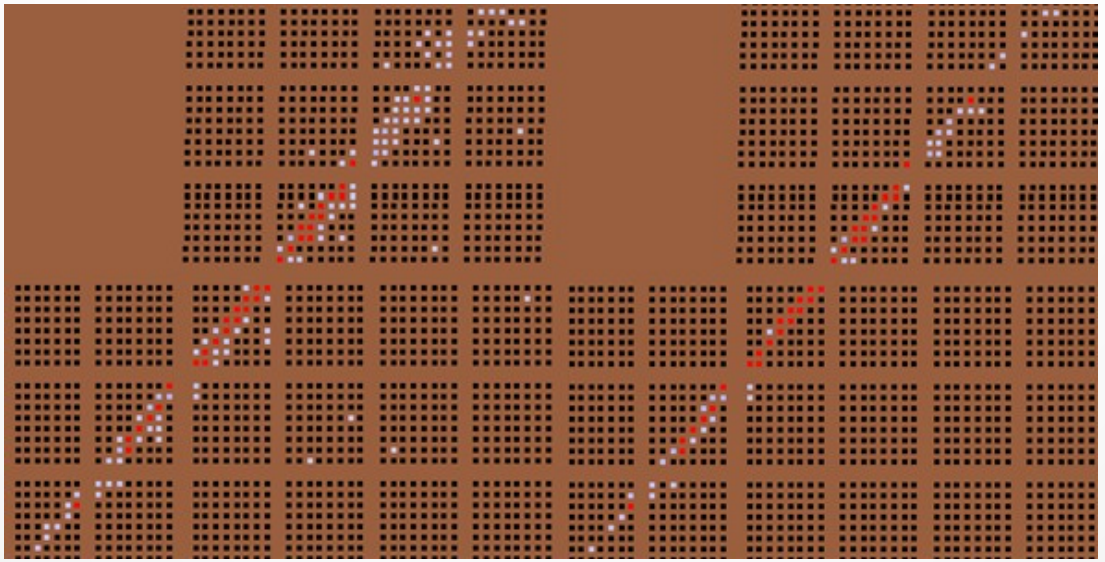






Example (2)



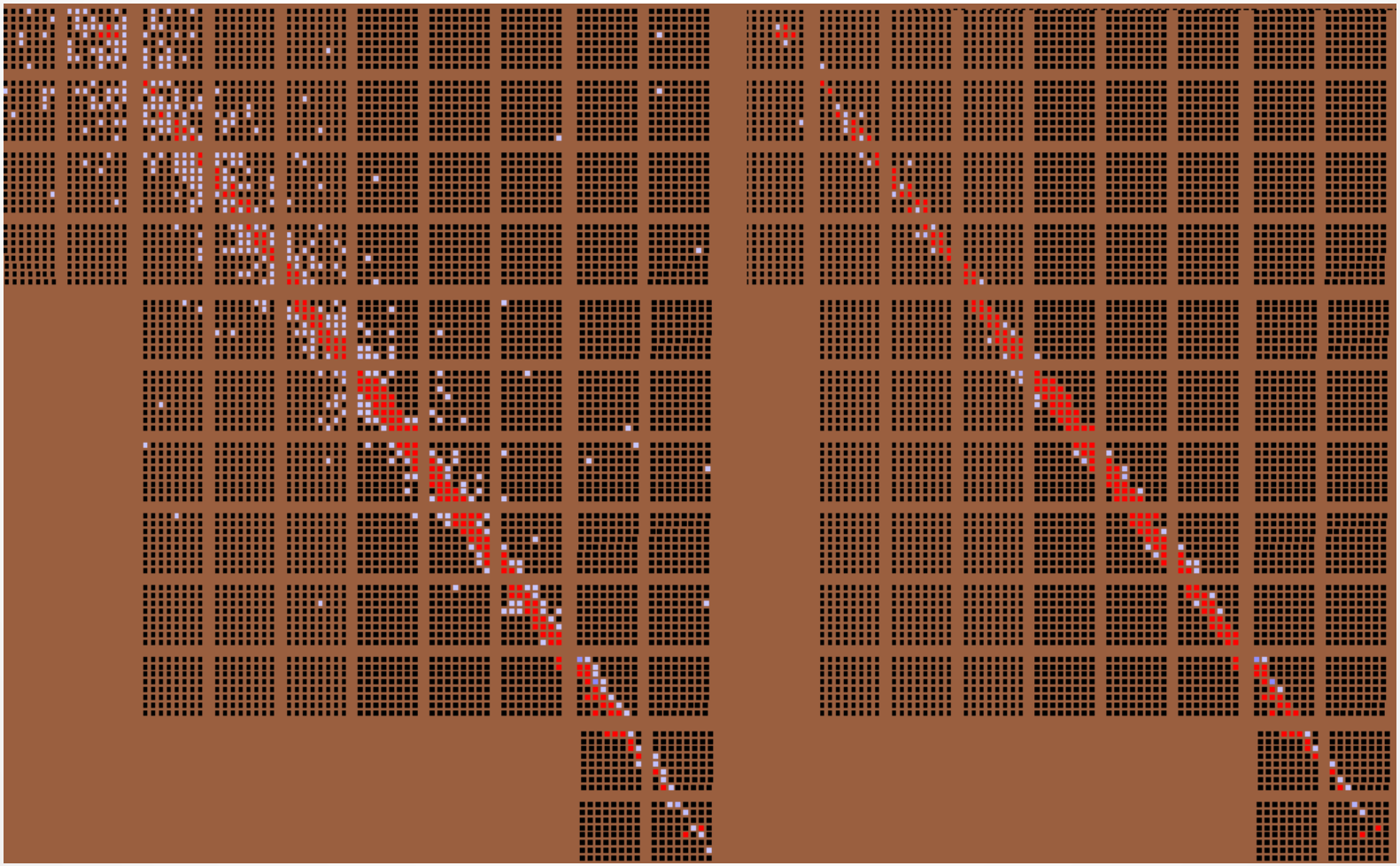


Example: angle 75, 1e20eV

parameters: dr = 1, threshold = 2

shower in the background: threshold = 0

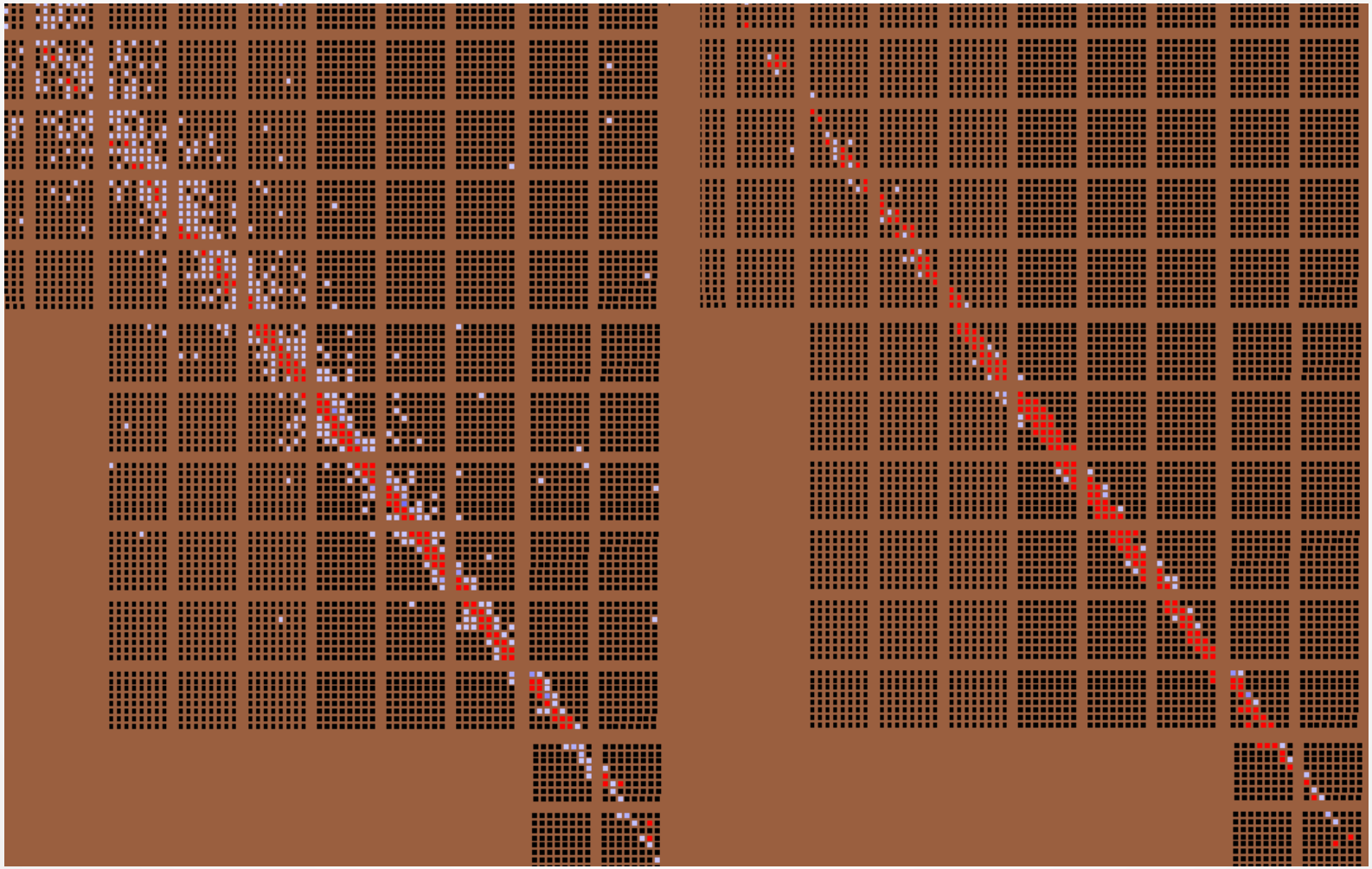
shower in the background: threshold = 1



Example: angle 75, 1e20eV

parameters: dr = 1.5, threshold = 2

shower in the background: threshold = 0 shower in the background: threshold = 1

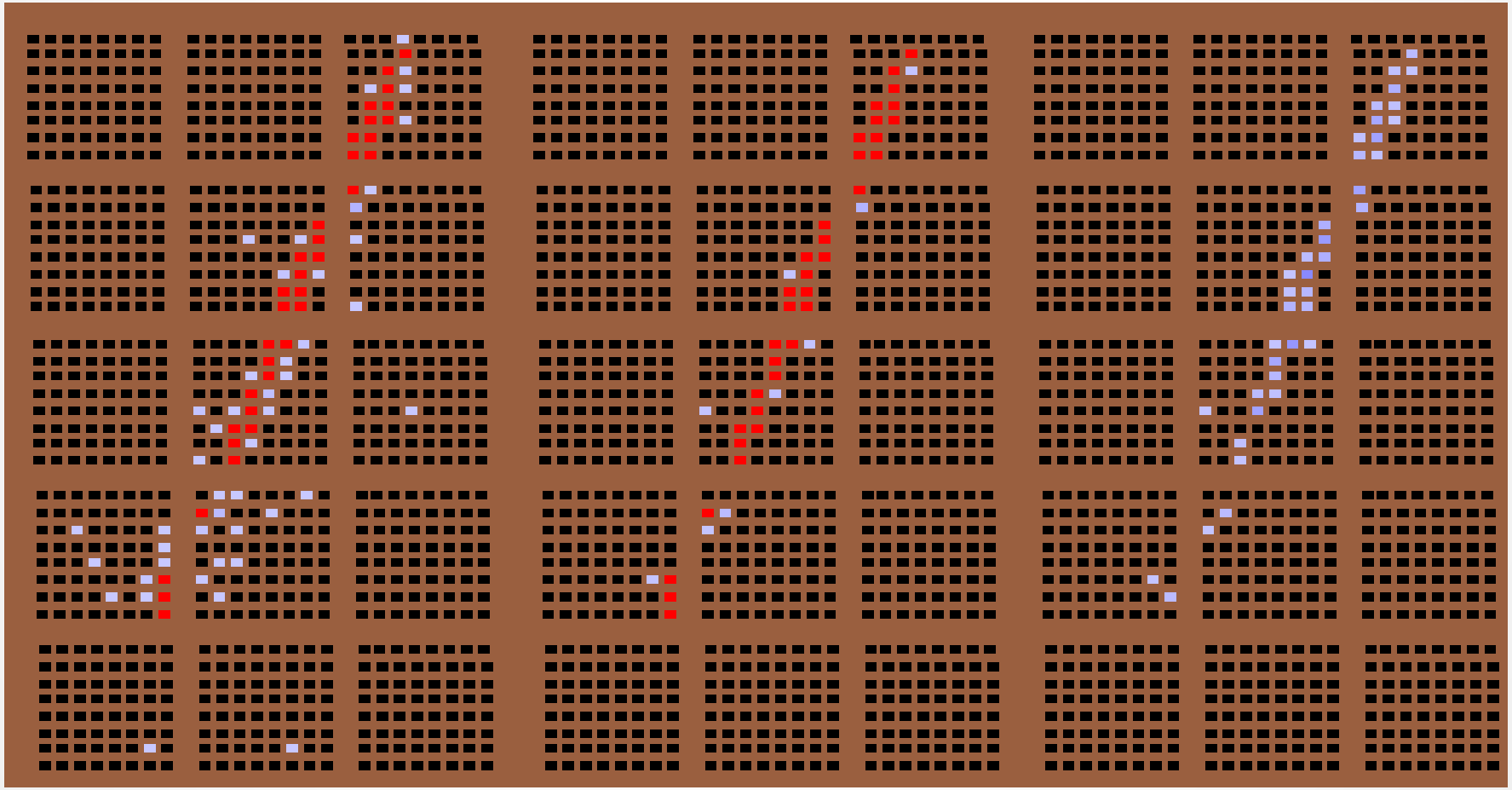


Example: angle 60, 1e20 eV

parameters: dr = 1, threshold = 2

shower in the background:

threshold = 0 threshold = 1 threshold = 1



Conclusions

Hough method was applied in two versions.

Method optimalization in progress

Comparsion with PWISE in next weeks

To be commited to ESAF Autumn