MAPS – Beam Test: tracking efficiencies

Also featuring: AM's Monte Carlo Beam Test Simulation

MAPS Group Meeting, RAL

Jamie Ballin with Paul Dauncey & Anne-Marie Magnan

HEP, Imperial College, London
j.ballin06@imperial.ac.uk

29th February 2008

Outline

1. Making χ^2 fits for real tracks
   - Overview
   - The wondrous libMapsTracks.so library
   - Aligning the system
2. Tracking results
   - Alignment and errors used
   - χ^2 probabilities
   - Beam profile
   - Residual on fourth sensor
3. Sensor efficiencies
   - Efficiency plots
   - x, y, t efficiency and inefficiency plots
   - Cross checks
   - AM's MC Simulation

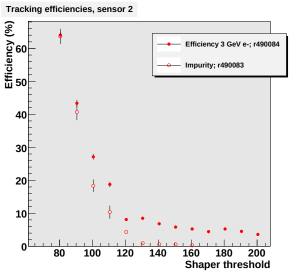
Last time...

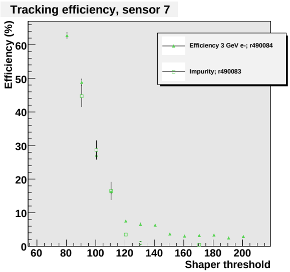
Defining a loose efficiency

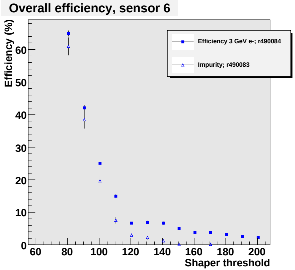
- For each bunch crossing, count how many hits each sensor has.
- For the sensors held at nominal, make a track when each of the 3 sensors has at least one hit. Get N tracks.
- Ask whether the threshold-scanned sensor confirms this. Get i confirmations, and $N - i$ rejections.
- Efficiency ϵ is simply,

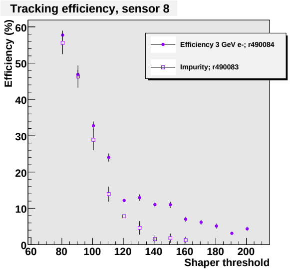
$$\epsilon = \frac{i}{N} \times 100\%$$

Controversial results









Queries over purity of sample

- Were we being thrown by fakes?
- Applying the same analysis to noise runs suggested that fakes were not dominating the efficiency curve.
- But impurity did grow alarmingly for low thresholds.

What constitutes a track?

- One hit in each of three or four layers at one particular bunch crossing.
- Now apply a χ^2 fit: Define χ^2 for one dimension (e.g. x) for N points as,

$$\chi_x^2 = \sum_{i=1}^{N} \left[x_i - (p_0 + z_i p_1)\right]^2 / \sigma_i^2 \qquad (1)$$

where p_j are the fit parameters (to be determined), and σ_i is the error intrinsic to the measurement at z_i, for z representing the experiment axis.

- Let us take $\sigma_i = \sigma_0$ (though I do not take $\sigma_x = \sigma_y$). Assume uncorrelated errors.
- Start by minimizing χ_x^2 for each track: you get a matrix equation,

$$\begin{pmatrix} N & \sum_i z_i \\ \sum_i z_i & \sum_i z_i^2 \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \end{pmatrix} = \begin{pmatrix} \sum_i x_i \\ \sum_i x_i z_i \end{pmatrix} \qquad (2)$$

- Invert to determine p_j

Evaluating the track quality

- Evaluate p_j for a given track, so the track in (x, y) is defined by

$$r = \begin{pmatrix} p_0 \\ q_0 \end{pmatrix} + z \begin{pmatrix} p_1 \\ q_1 \end{pmatrix} \tag{3}$$

  for q_j the fit parameters in y
- Compute χ^2
- Use `TMath::Prob(chisq, ndf)` to evaluate probability that the track is real and not formed through statistical fluctuations. Here `ndf` = number of points $N - 2$ for the 2 p_j.
- Currently evaluate x and y seperately, but one can easily combine them:

$$\chi^2_{\mathrm{tot}} = \chi^2_x + \chi^2_y \tag{4}$$

- Finally, define θ_z as the polar angle between the track and the z axis,

$$\cos\theta_z = \frac{1}{\sqrt{p_1^2 + q_1^2 + 1}} \tag{5}$$

Introducing `MapsTrack`, `MapsSensor` and `MapsTrackManager`

New and exciting code structure, completely independent of `MpsAnalysis` and DAQ framework

- MapsSensor: is aware of its *z* positioning and id.
  - Is told by users whether it's been efficient or not.
  - Is also told of residuals for alignment.
  - Application code queries sensors for their plots.
- MapsTrack: The major workhorse.
  - Holds an STL `map` of `MapsSensor*`s and `std::pair<int, int>`s.
  - Provides methods for evaluating fit parameters and such like.
  - Must hold either 3 or 4 hits; behaviour is undefined for anything else.
- MapsTrackManager: Utility class used to persist tracks. You create a set of `MapsSensor*`s and `MapsTrack*`s, and add them to an instance of this object. One can then use,
  - `exportToRootFile`: saves `MapsTrack`s and `MapsSensor`s to a ROOT file using `TTree` structure
  - `recreateFromRootFile`: recreates tracks and sensors from ROOT file

Flexible code

- The only truly persistent state is a track's hits: all other quantities are reevaluated at run-time! This is uber-flexible!
- Very easy to reapply new track fit errors
- Trivial to apply new alignments!
- Idea is to use/write little tools to read in tracks, change parameters, and re-export them to a new ROOT file

Determining alignment

Physical (x, y, z) in mm in World coordinate will not directly correspond to pixel (x, y, z)

- DESY beam test set up for tracking runs: z coordinates[1] for each sensor taken as,
  - 8: 0 mm
  - 7: 18 mm
  - 2: 36 mm
  - 6: 54 mm
- Let sensors 8 and 6 define a World coordinate system

[1] While these may be slightly out, what matters is that they were evenly spaced

Determining alignment

- When one has a track with sensors 6 and 8 containing hits, extrapolate[2] their hits to sensors 7 and 2: plot the residual, r_x, defined as

$$r_x = x_{\text{projected}} - x_{\text{real}} \tag{6}$$

  similarly for r_y.
- Expect a peak for real track hits, and a uniform background for noise.
- It can be shown that the width σ_{fit} of the fitted gaussian has the correspondance,

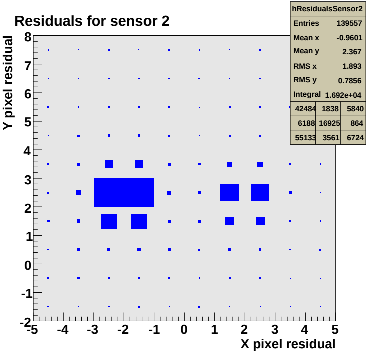
$$\sigma_0 = 1.25\sigma_{\text{fit}} \tag{7}$$

  for σ_0 the intrinsic resolution of the sensor.

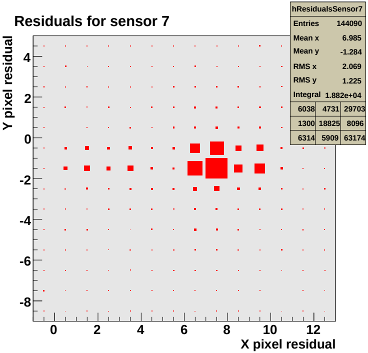
[2] A literal $x = mz + c$ baby line defined by the two points, not a χ^2 fit

Alignment results

Ghosting in the *x* coordinate ($\sim$ 10% level)? Fit to main peak only...
All units in pixels



$r_{x,y|2} = (-2.0 \pm 0.6, 2.3 \pm 0.5)$



$r_{x,y|7} = (7.3 \pm 0.7, -1.3 \pm 0.6)$

Alignment and errors adopted

- From previous slide,

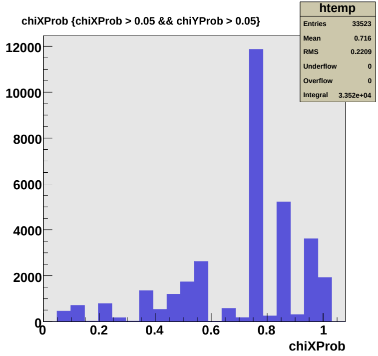
$$r_{x,y|2} = (-2.0 \pm 0.6, 2.3 \pm 0.5)$$
$$r_{x,y|7} = (7.3 \pm 0.7, -1.3 \pm 0.6)$$

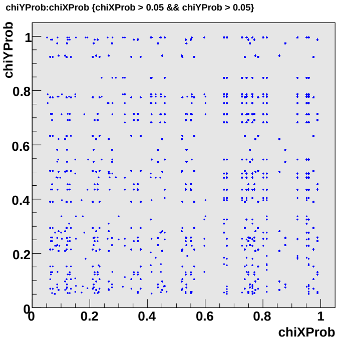
- Add these values to all hits for sensors 2 and 7 when fitting anything. All results from here onwards are in the aligned system.
- Using $\sigma_0 = 1.25\sigma_{\mathrm{fit}}$, take the error $e_{x,y}$ in mm as,

$$e_{x,y} = (0.0438, 0.0350) \quad (8)$$

CALICE Calorimeter for ILC

χ^2 probabilities

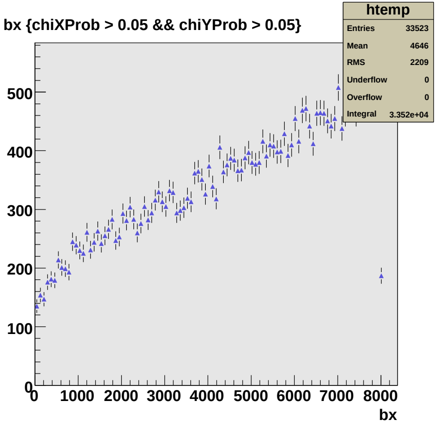




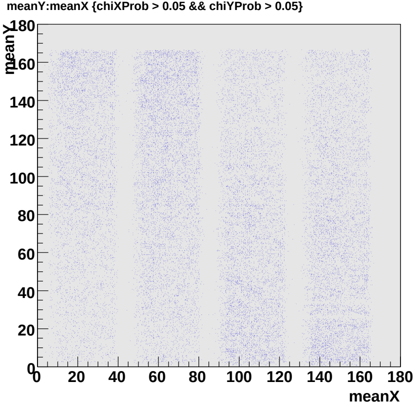
- Cut: $p_x \& p_y > 0.05$ very effectively excludes noisy tracks
- p_x and p_y distributed over entire interval, but with a slight bias to high probability: might do well to tighten σ_0, but I wanted to be objective.
- Beware: discrete system causes discretisation in p_x, p_y for common track combinations!

Bunch crossing number of all good tracks

Interesting. . .



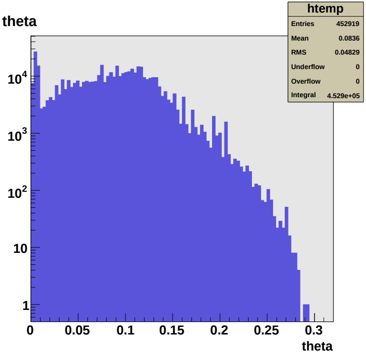
$\bar{x}, \bar{y}$ all track hits



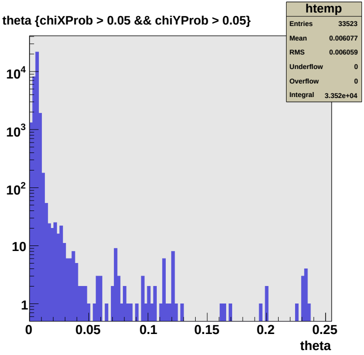
Apologies: The result presented at the last meeting in my absence was erroneous, and showed a beam spot (which does not exist here) which was born solely out of a phase space effect, and with no proper cuts on track quality.

θ_z of tracks

No track quality cuts

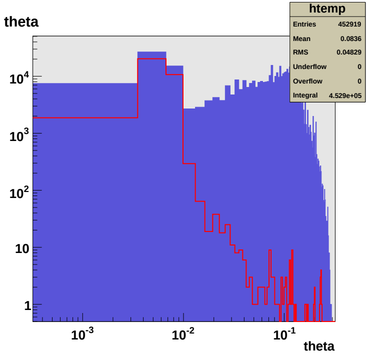


Applying p_x, p_y cut



θ_z of tracks

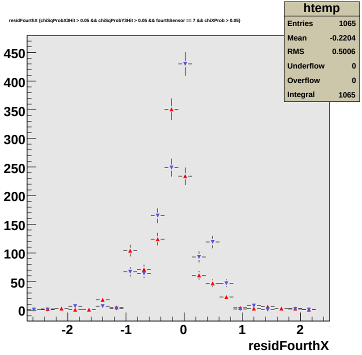
Or, put it another way



- Clearly low θ_z implies beam particles and high p_x, p_y.
- Not useful for cosmics
- Don't use as an explicit cut

Fourth hit residual

- When $\exists$ a fourth hit, use the other three hits to define a new track.
- Check this three hit track passes the usual cuts.
- Find the residual between the extrapolated track and the fourth sensor's hit.
- Use this as a further cut on the fourth sensor's efficiency.
- We observe consistent alignment!

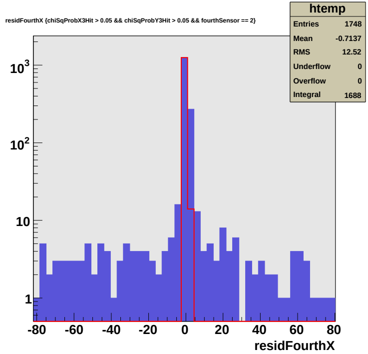


Sensor 2, Sensor 7

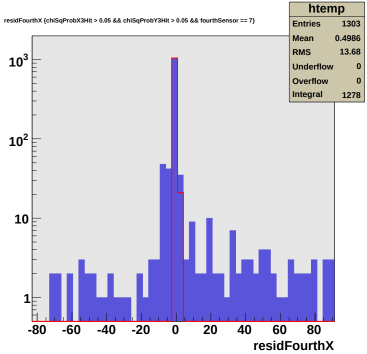
Fourth hit resiual

Applying cut on four track p_x, p_y given a good three hit track p_x, p_y is very effective

Sensor 2

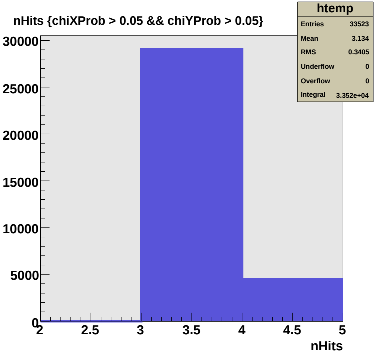


Sensor 7



- No cut on four hit track probabilities
- Cut applied

Crude efficiency calculation



Sanity check suggests, before proceeding further:

$$\frac{n_4}{n_3} = \frac{5000}{28,000} \sim 17\% \qquad (9)$$

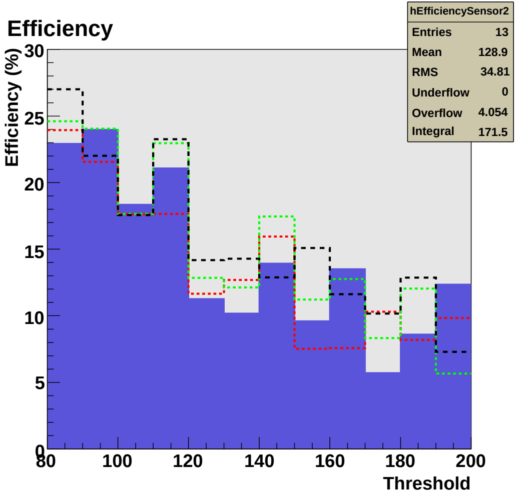
(That's with cuts applied too)

Sensor efficiencies

Selection criteria

- Require 3 hit track $p_x \& p_y > 0.05$
- Efficient if $\exists$ hits and 4 hit track has $p_x \& p_y > 0.05$
- Inefficient $\nexists$ 4th hit

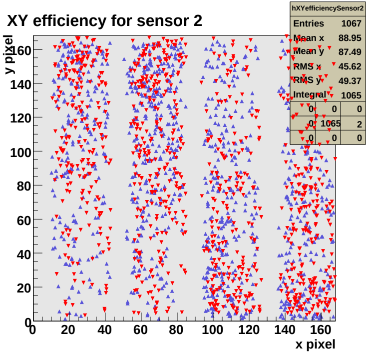
Efficiency with threshold



Where were we efficient?

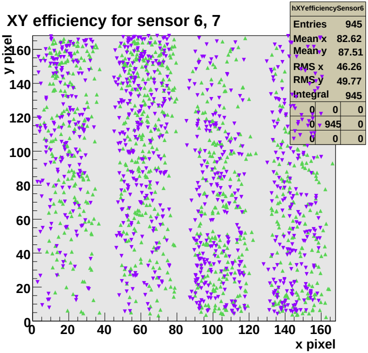
- Hints of an optimal capacitor layout?

Rotated sensors

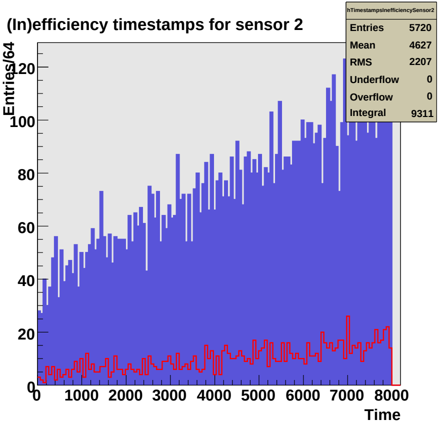


Rotated ⇒ samplers on the left, shapers on the right

Non–rotated sensors



When were we efficient?



- Efficient
- Inefficient

Raw efficiencies

```
Sensor id 2 [z=36, al=(-2.015, 2.347)]:
Efficiency:
Thresh:  80 :  22.93
Thresh:  90 :  23.98
Thresh:  100 :  18.36
Thresh:  110 :  21.08
Thresh:  120 :  11.27
Thresh:  130 :  10.19
Thresh:  140 :  13.94
Thresh:  150 :  9.6
Thresh:  160 :  13.51
Thresh:  170 :  5.714
Thresh:  180 :  8.607
Thresh:  190 :  12.36
Sensor id 6 [z=54, al=(0, 0)]:  Efficiency:
Thresh:  80 :  23.94
Thresh:  90 :  21.56
Thresh:  100 :  17.61
Thresh:  110 :  17.65
Thresh:  120 :  11.66
Thresh:  130 :  12.69
Thresh:  140 :  15.96
Thresh:  150 :  7.527
Thresh:  160 :  7.576
Thresh:  170 :  10.31
Thresh:  180 :  8.187

Thresh:  190 :  9.827
```

```
Sensor id 7 [z=18, al=(7.3, -1.25)]:
Efficiency:
Thresh:  80 :  24.6
Thresh:  90 :  24.05
Thresh:  100 :  17.65
Thresh:  110 :  22.95
Thresh:  120 :  12.84
Thresh:  130 :  12.12
Thresh:  140 :  17.46
Thresh:  150 :  11.22
Thresh:  160 :  12.77
Thresh:  170 :  8.333
Thresh:  180 :  12.04
Thresh:  190 :  5.66
Sensor id 8 [z=0, al=(0, 0)]:  Efficiency:
Thresh:  80 :  27.01
Thresh:  90 :  22.01
Thresh:  100 :  17.56
Thresh:  110 :  23.26
Thresh:  120 :  14.19
Thresh:  130 :  14.29
Thresh:  140 :  12.88
Thresh:  150 :  15.09
Thresh:  160 :  11.61
Thresh:  170 :  10.17
Thresh:  180 :  12.87

Thresh:  190 :  7.292
```

Noise–only run and “displaced time”

Cross checks

- Noise only runs: effectively 0% efficient, with few 3 hit tracks passing selections
- Run 490083 (noise) – 252k bunch trains

```
ExtractEfficiencies:  summary:
Total candidate tracks:  20
Efficient hits:  0
Inefficient hits:  20
```

- Contamination in beam run? Beam run has 1.3 M bunch trains $\Rightarrow 20/252,000 \times 1.3 \times 10^6 = 103$ of the 3 hit tracks in the beam run are fakes
- Just $103/34,948 \times 100\% = 0.3\%$ of three hit tracks are fakes
- Using a decorrelated time for the fourth hit $\Rightarrow$ no four hit tracks passing either!

Consistency checks

Run 490084 (beam) – 1.3M bunch trains

```
ExtractEfficiencies: summary:
Total candidate tracks: 34948
Efficient hits: 4459
Inefficient hits: 30489
```

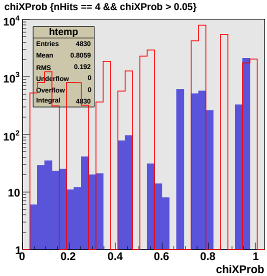
- Cross check the efficiency with,

$$\frac{< n_4 >}{n_{\text{candidates}}} = \epsilon^4 \tag{10}$$

- $\frac{<n_4>}{<n_3>} = \frac{4459}{34,948} = 12.8\%$
- Expect $\sim 1 - 10e^-$ per bunch train (poisson distribution) $\Rightarrow (\frac{4459}{5 \times 1.3 \times 10^6})^{\frac{1}{4}} = 16.2\%$ efficient

Cuts

- Could the fourth hit cut be too tight?



(3 hits, 4 hits)

- If anything, the p_x, p_y are probably too loose.
- Furthermore $\frac{<n_4>}{<n_3>}$ is stable at the order of 1% point if I increase the p_x, p_y cut to 0.1, and reduce σ_0 by 25%.

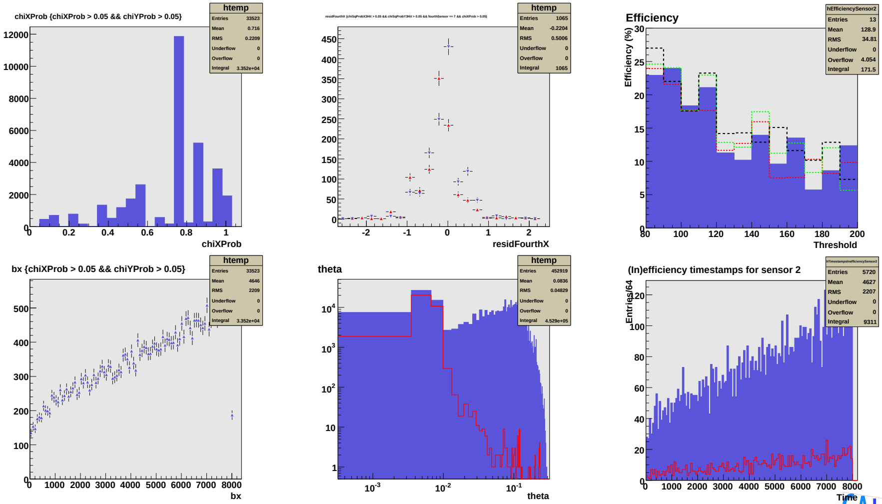
Summary

We have a reliable and stable way of finding tracks

- Alignment is stable
- Track counts are not very sensitive to cuts and error specifications
- Sadly the efficiency remains sub 20%
- Can even find tracks at thresholds < 100, but we can't operate the sensor like this(?)

You are welcome to try it for yourself with libMapsTracks.so…

Summary


chiXProb {chiXProb > 0.05 && chiYProb > 0.05}
htemp
Entries 33523
Mean 0.716
RMS 0.2209
Underflow 0
Overflow 0
Integral 3.352e+04
chiXProb
htemp
Entries 1065
Mean -0.2204
RMS 0.5006
Underflow 0
Overflow 0
Integral 1065
residFourthX
Efficiency
hEfficiencySensor2
Entries 13
Mean 128.9
RMS 34.81
Underflow 0
Overflow 4.054
Integral 171.5
Efficiency (%)
Threshold
bx {chiXProb > 0.05 && chiYProb > 0.05}
htemp
Entries 33523
Mean 4646
RMS 2209
Underflow 0
Overflow 0
Integral 3.352e+04
bx
theta
htemp
Entries 452919
Mean 0.0836
RMS 0.04829
Underflow 0
Overflow 0
Integral 4.529e+05
theta
(In)efficiency timestamps for sensor 2
Entries 5720
Mean 4627
RMS 2207
Underflow 0
Overflow 0
Integral 9311
Entries/64
Time

Anne–Marie's Monte Carlo Simulation

- Beam test setup

MC Simulation

- 4 layers simulation, equivalent to run 490084,
- no noise only, but seing Jamie's cuts in space is removing them completely anyway, we can neglect them...
- noise added to the signal
- complete charge spread with last results from Giulio (full deep-pwell+nwell simulation in the centre pixel)
- No shapers/samplers consideration: all done with shapers. On the point of view of the simulation, the samplers have *2 in gain, but then *2 in nominal threshold, so converting back to eV gives back the same factor $\Rightarrow$ would just have a slightly higher noise (estimated $\sim$ 30% higher : noise proba $\sim 7 \times 10^{-6}$ 300 TU).

Estimation of the noise

- A quick calculation gives : 120 TU (threshold units),
  10^{-6} noise proba = 4.75 σ (just from a gaussian distribution)
  $\Rightarrow$ 1 $\sigma \sim$ 25 TU.
- First estimation from old σ(E)/E vs thresh plot: 3 keV $\Rightarrow$ 25% efficiency.
- and 25% efficiency is seen 100 TU,
- 100 TU has noise proba $\sim$ 4 σ,
- so 4 σ=3 keV gives 1 σ = $\sim$750 eV.

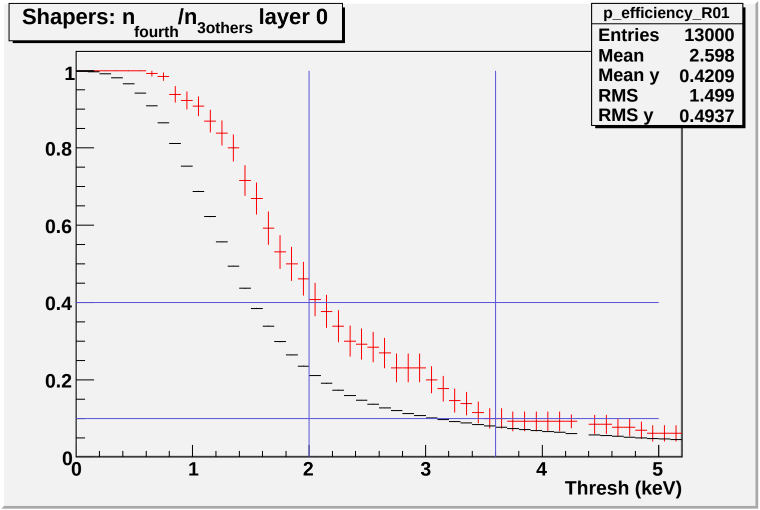
Results

- Simulation hence done with 750 noise (red curve),
- and ∼3.6 keV (∼120 TU) nominal threshold for 3 sensors, threshold scan between 0 and 10 keV on the fourth sensor.
- For comparison, black curve=80 eV noise (what would be expected if the nominal threshold chosen at 120 TU was compatible with half a MIP in the worse case = ∼130 electrons *3.2 ∼ 380 eV = 4.75 * noise)
- Compatibility with data:

eff (%)	*dataTU*	*MCkeV*	conv factor keV/TU
25	85	2.7	0.032
20	105	3.0	0.028
15	145	3.3	0.023
10	200	3.6	0.018

- so not the same shape, but roughly 0.03 *25 = 0.75 keV noise ⇒ consistent.
- if it was as expected: the efficiency at nominal threshold would be better than 99% ...

Plottage


Shapers: $n_{fourth}/n_{3others}$ layer 0
p_efficiency_R01
Entries 13000
Mean 2.598
Mean y 0.4209
RMS 1.499
RMS y 0.4937
1
0.8
0.6
0.4
0.2
0
0
1
2
3
4
5
Thresh (keV)