Radial Smearing

Simple radial energy distribution in tower grids

**Ignore longitudinal development**
- Particle energy distributed transverse to direction of flight of particle in a plane through the particle impact point into the calorimeter
- Shape of distribution from experiment/full simulation

**Integrated energy in profile is the same as particle energy**
- No calibration/acceptance/smearing of energy

**Distributed energies projected into regular eta/phi grid within modeled detector acceptance**
- Fakes calorimeter tower signal definition, including high eta losses
- Different grid and eta acceptance for EM particles
Radial Energy Projection
Toy detector defaults

- **EM acceptance** \(-2.5 < \eta < 2.5\)
  - Photons/electrons outside are mapped onto HAD towers
- **HAD acceptance** \(-5.0 < \eta < 5.0\)
  - Particles outside are ignored completely

**Cylindrical calorimeter**
- \(R = 1200 \text{ mm, } -2500 \text{ mm} < z < 2500 \text{ mm}\)
- **High granularity**
  - 0.025 x 0.025 (EM)
  - 0.1 x 0.1 (HAD)

**Shower shapes**

- **Started with Gaussian within cylinder**
  - Lateral extend /cylinder radius 80 mm for EM particles (also in HAD grid!)
  - 160 mm for all others
  - Gaussian showers are too wide, this was just a simplification!

- **Now experimental electromagnetic and hadronic shower shapes**
  - H1 data easily available (with some help) – representative for ATLAS (CMS?)-like calorimeters
  - Energy distributed in small “spots”
Points/open histogram: data + simulated signal with noise cuts

Shaded histogram: simulated shower development, no noise or cuts

5 GeV $e^-$

30 GeV $e^-$

PL Thesis, 1992
Points/solid line histogram: data + simulated signal with noise cuts

Dashed line histogram: simulated shower development, no noise or cuts
Two files needed:
DetectorModel.h and DetectorModel.cxx
Uses fastjet/PseudoJet.h and STL – nothing else!
Compile DetectorModel.cxx

Usage (default):
#include “DetectorModel.h”
...
// default smearing module
DetectorModel::RadialSmearing smear;
... fill std::vector<PseudoJet> input with final state particles and set
the user_index of each of those to the particle pdg code!!
// retrieve pseudo-particles after smearing
const std::vector<PseudoJet>& smeared = smear.smear(input);
// do your thing...

Observations:
Expect about 20 x more particles than in the generator final state
Topology dependent, here Pythia QCD di-jets
Slow execution
Needs much more optimization
Pythia8 QCD Di-jets

log(E_t)
Input Shapes Fits To Data

\[
\frac{dE}{E}(r)
\]

**EM**

**Gauss**

**HAD**

**Gauss**
$$\frac{1}{E} \int_{0}^{r} dE(r') dr'$$
Some energy dependence observed

Not very significant
Implemented it anyway
Only two profiles used
E = 30, 120 GeV

Log-like interpolation between those

\[ S(E, r) = S(30 \text{ GeV}, r) \cdot f(E, r) \]

\[ = (a(r) + b(r) \cdot \log(E/\text{GeV})) \]

99% containment range
\( r_{\text{max}} \) (mm)

\[
\begin{align*}
\log_{10}(E/\text{GeV}) & \\
\log_{10}(dE/E) & \\
\end{align*}
\]

\( (r_{\text{max}} = 80 \text{ mm for } E > 200 \text{ GeV}) \)

(not normalized!)