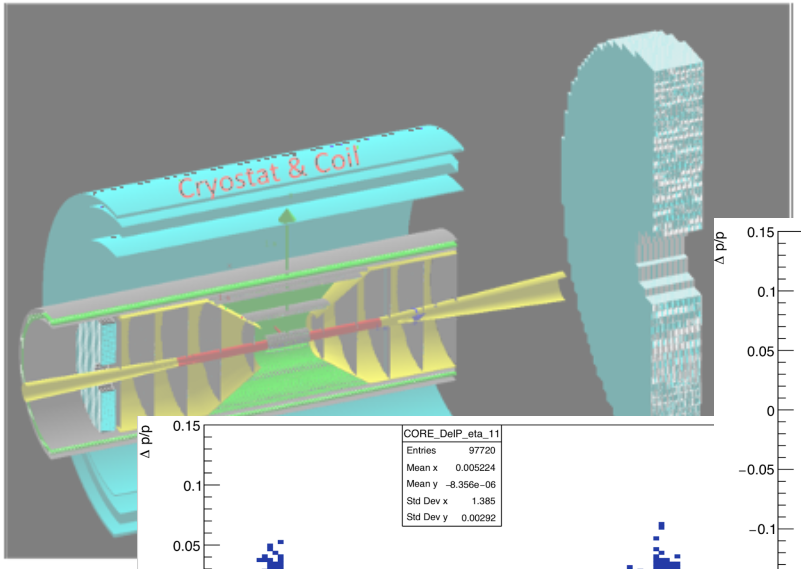
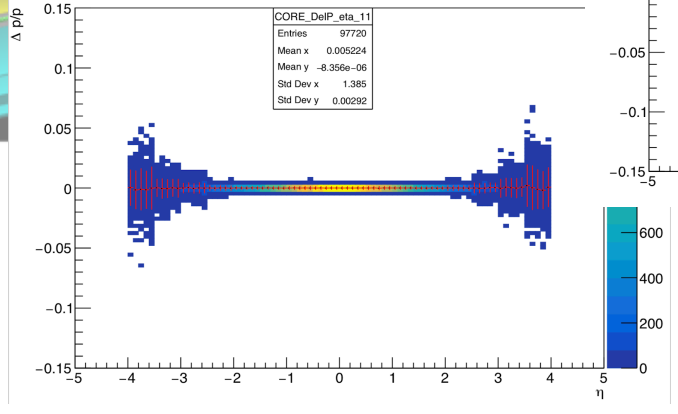
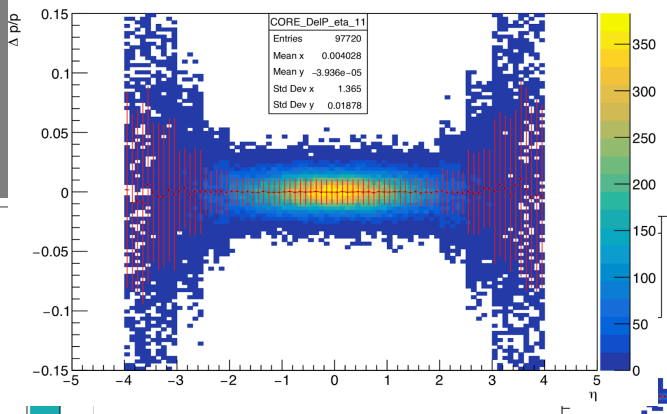
Kolja Kauder

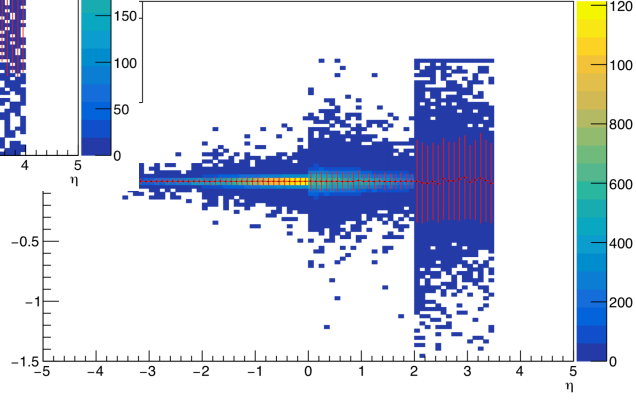
BROOKHAVEN NATIONAL LABORATORY



CORE in Eic-Smear







U.S. DEPARTMENT OF ENERGY | Office of Science

CORE Meeting
Online, May 2021

How to Install Eic-smear and Generators

Don't

Just use

```
$ setenv EIC_LEVEL dev
$ source /cvmfs/eic.opensciencegrid.org/x8664_sl7/MCEG/releases/etc/eic_cshrc.csh
```

Or the bash version

```
$ export EIC_LEVEL=dev
$ source /cvmfs/eic.opensciencegrid.org/x8664_sl7/MCEG/releases/etc/eic_bash.sh
```

- Works at BNL, JLab, most likely many other places right away
- Otherwise, use singularity as a precursor. fun4all works!
- Contact me for larger scale job submission (works at labs, OSG)
- ESCalate, spack, docker, …
- If you must, local install is easy for eic-smear, generators will need more packages (cernlib, lhapdf, …)
    - Big Three: Best to use their containers - ask Markus Diefenthaler!

How to Install and Use your own Detectors

More instructions at eicsmeardetectors

```
$ echo $MYEICSOFTWARE # Adapt to your needs
/home/coreuser/software
$ git clone https://github.com/eic/eicsmeardetectors.git
$ cd eicsmeardetectors; mkdir build; cd build
$ cmake -DBUILD_TESTS=ON -DCMAKE_INSTALL_PREFIX=$MYEICSOFTWARE ..
$ make install
```

- Important: Adapt PATH and LD_LIBRARY_PATH:

```
bash $ export LD_LIBRARY_PATH=$MYEICSOFTWARE/lib:$LD_LIBRARY_PATH
bash $ export PATH=$MYEICSOFTWARE/bin:$PATH
tcsh $ setenv LD_LIBRARY_PATH $MYEICSOFTWARE/lib:$LD_LIBRARY_PATH
tcsh $ setenv PATH $MYEICSOFTWARE/bin:$PATH
## On a Mac, use DYLD_LIBRARY_PATH instead
```

- eic-smear will show that you're doing it right

```
$ eic-smear
/Users/kkauder/software/lib/libeicsmear.dylib
/Users/kkauder/eicdev/eicsmeardetectors/build/libeicsmeardetectors.dylib
```

Transformation

```
$ root -l
root [0] gSystem->Load("libeicsmear")
root [1] BuildTree("pythia.txt.gz",".",-1,"log.txt")
```

Or compile:

```
$ g++ `root-config --cflags` \
`root-config --libs` \
-I<…>/include -L<…>/lib -leicsmear …
```

```
#include <TSystem.h>
#include <eicsmear/functions.h>
int main(){
  gSystem->Load("libeicsmear");
  BuildTree("largepythia.txt", ".", -1);
  return 0;
}
```

Accepted Generators:

PYTHIA6 | DPMJet | PEPSI | Sartre | Djangoh | Milou | Rapgap | BeAGLE

Any HepMC: PYTHIA8 eSTARLIGHT HERWIG, SHERPA, …

Interactive use

- Automatic loading of libraries and version information with small wrapper

```
$ eic-smear
Using eic-smear version: 1.1.2
Using these eic-smear libraries :
/Users/kkauder/software/lib/libeicsmear.dylib
/Users/kkauder/software/lib/libeicsmeardetectors.dylib
eic-smear [0] BuildTree("pythia.txt",".", -1, "log.txt")
```

```
Replaces:
root [0] gSystem->Load("libeicsmear");
root [1] gSystem->Load("libeicsmeardetectors")
```

- Load the script or use shortcut function

```
eic-smear [1] .L SmearCore_0_1.cxx
eic-smear [2] auto d = BuildCore_0_1 (2.0);
# or
eic-smear [1] auto d = BuildByName("core", 2.0)
```

New: Specify B-field

Now part of the repo

https://github.com/eic/eicsmeardetectors/blob/master/SmearCore_0_1.cxx

- All in one directly from the shell:

```
$ echo 'SmearTree(BuildByName("core",2), "pythia.root")' | eic-smear
```

Analysis

```
46   TChain* inTree = new TChain("EICTree");
47   inTree->Add(inFileName1);
48   inTree->AddFriend("Smeared",inFileName2);
49
50   // Setup Input Event Buffer
51   erhic::EventPythia* inEvent(NULL);     inTree->SetBranchAddress("event",&inEvent);
52   Smear::Event* inEventS(NULL);          inTree->SetBranchAddress("eventS",&inEventS);
```

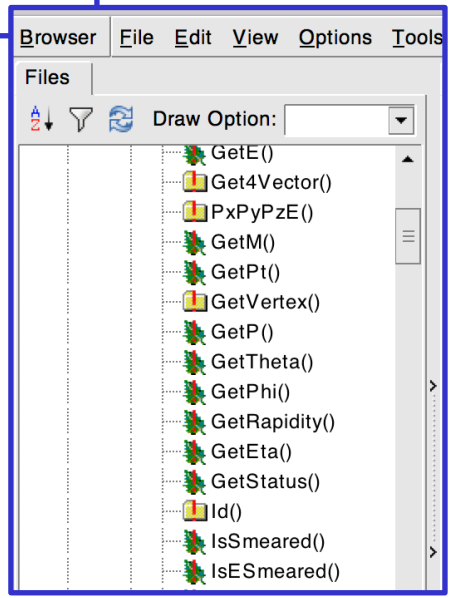
Befriend and get matched branches

```
124       const Smear::ParticleMCS* inParticleS = inEventS->GetTrack(j); // Smeared Particle
125       const Particle* inParticle = inEvent->GetTrack(j); // Unsmeared Particle
```

Particle Loop

- Access properties:

```
if ( inParticleS->IsPSmeared() ){
  auto delP = (inParticle->GetP() - inParticleS->GetP()) / inParticle->GetP();
  coll.DelP_th->Fill(inParticle->GetTheta(), delP);
  coll.DelP_eta->Fill(inParticle->GetEta(), delP);
}
```



Provided QA programs demonstrates usage

https://github.com/eic/eicsmeardetectors/blob/master/tests/qaplots.cxx

https://github.com/eic/eicsmeardetectors/blob/master/tests/pgunqa.cxx

CORE: Angular resolutions

```
79    // Perfect phi and theta for all particles
80    // ------------------------------------
81    // total coverage for tracker, ecal, and hcal is -4.0 < eta < 4.0
82    Smear::Acceptance::Zone AngleZoneCommon(ThetaFromEta ( 4.0 ),ThetaFromEta ( -4.0 ));
83    Smear::Device SmearThetaCommon(Smear::kTheta, "0.0");
84    SmearThetaCommon.Accept.AddZone(AngleZoneCommon);
85    SmearThetaCommon.Accept.SetGenre(Smear::kAll);
86    det.AddDevice(SmearThetaCommon);
87
88    Smear::Device SmearPhiCommon(Smear::kPhi, "0.0");
89    SmearPhiCommon.Accept.AddZone(AngleZoneCommon);
90    SmearPhiCommon.Accept.SetGenre(Smear::kAll);
91    det.AddDevice(SmearPhiCommon);
```

What you may want to change:

- More realistic values – including p_T, η, … dependence
- Separate values for separate detectors – use `SetGenre()`

CORE: Tracking

Parameterization source

```
97    // |eta|
98    const std::valarray<double> eta_min = {   0,   0.5,     1,   1.5,     2,   2.5,     3,   3.5 };
99    const std::valarray<double> eta_max = { 0.5,     1,   1.5,     2,   2.5,     3,   3.5,     4 };
100   // sigma_p/p ~ A% * p + B%
101   const std::valarray<double> A3T   = { 0.018, 0.016, 0.016, 0.012, 0.018, 0.039, 0.103, 0.295 };
102   const std::valarray<double> B3T   = { 0.369, 0.428, 0.427, 0.462, 0.719, 1.336, 2.428, 4.552 };
103   const std::valarray<double> A1p4T = { 0.038, 0.035, 0.035, 0.026, 0.041, 0.088, 0.217, 0.610 };
104   const std::valarray<double> B1p4T = { 0.816, 0.898, 0.921, 0.997, 1.548, 2.830, 5.234, 9.797 };
105
106   const std::valarray<double> A = CalcA ( Bfield, 1.4, A1p4T, 3.0, A3T );
107   const std::valarray<double> B = CalcB ( Bfield, 1.4, B1p4T, 3.0, B3T );
108   AssembleCoreTracker ( det, eta_min, eta_max, A, B );
```

Interpolate to any B-field

```
265   std::valarray<double> CalcB ( const double Bfield,
266                                 const double x1, const std::valarray<double> B1,
267                                 const double x2, const std::valarray<double> B2 ){
275     // Basic linear interpolation
276     // f ( x ) = f(x1) + ( x-x1 ) * ( f(x2) - f(x1) ) / (x2 - x1 )
277     return B1 + (Bfield - x1 ) * (B2 - B1) / (x2 - x1 );
278   }
```

CORE: Assemble tracker

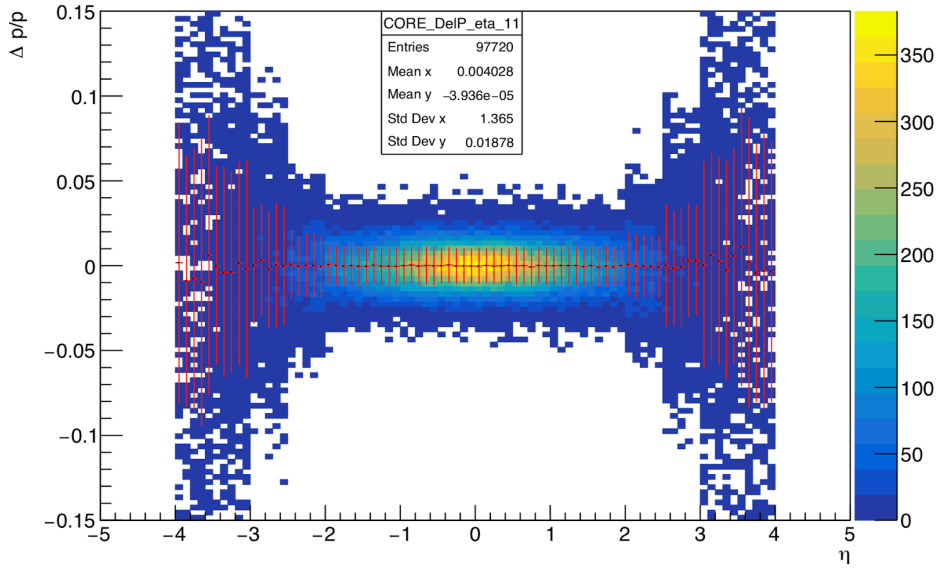
```
285    // sigma_p/p ~ A% * p + B %
286    const TString SmearString = "sqrt( pow (  AAA *P*P, 2) + pow ( BBB * P, 2) )";
287
288    for( auto i = 0; i< A.size(); ++i)  {
289      auto CurrentString = SmearString;
290      TString AAA = ""; AAA += 0.01*A[i];
291      TString BBB = ""; BBB += 0.01*B[i];
292      CurrentString.ReplaceAll ( "AAA", AAA );
293      CurrentString.ReplaceAll ( "BBB", BBB );
294
295      Smear::Device TrackPos(Smear::kP, CurrentString);
296      TrackPos.Accept.SetCharge(Smear::kCharged);
297      Smear::Device TrackNeg = TrackPos;
298
299      // Two zones, acceptance is symmetrical in eta
300      Smear::Acceptance::Zone TrackZonePos( ThetaFromEta (  eta_max[i] ), ThetaFromEta (  eta_min[i] ));
301      Smear::Acceptance::Zone TrackZoneNeg( ThetaFromEta ( -eta_min[i] ), ThetaFromEta ( -eta_max[i] ));
302
303      TrackPos.Accept.AddZone(TrackZonePos);
304      TrackNeg.Accept.AddZone(TrackZoneNeg);
305      det.AddDevice(TrackPos);
306      det.AddDevice(TrackNeg);
```

What you may want to change:

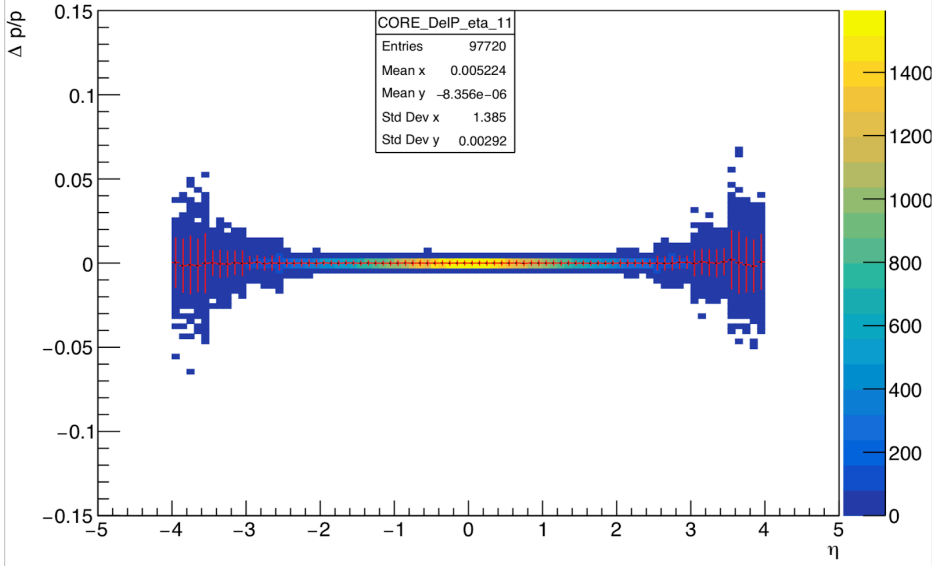
- parameters
- smooth 2+D fit

Testing tracking performance

Δp / p electrons
CORE @ 1T



Δp / p electrons
CORE @ 4T



CORE: Calorimetry (Ex.)

```
227     // Forward
228     // eta = 1 -- 3.5
229     // E>500 MeV not used because it suppresses smeared-up particles
230     // stoch. = 35%, const = 2% (aspirational goal)
231     Smear::Acceptance::Zone HcalFwdZone(ThetaFromEta ( 3.5 ),ThetaFromEta ( 1 ));
232     Smear::Device HcalFwd(Smear::kE, "sqrt(pow( 0.02*E, 2) + pow ( 0.35,2) *E)");
233     HcalFwd.Accept.AddZone(HcalFwdZone);
234     HcalFwd.Accept.SetGenre(Smear::kHadronic);
235     det.AddDevice(HcalFwd);
```

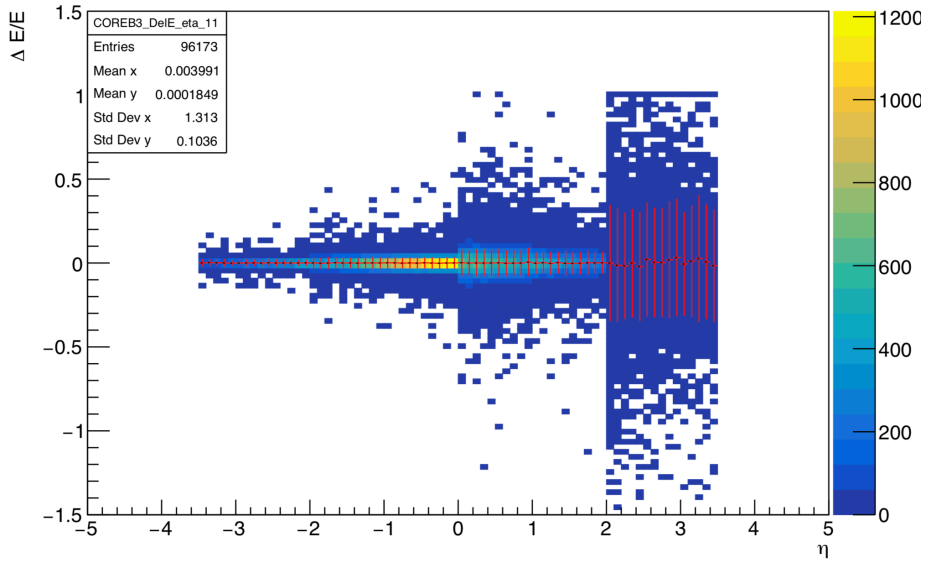
```
187     // Forward inner - Shashlyk guess
188     // eta = 2 -- 3.5
189     // E > 50 MeV not used because it suppresses smeared-up particles
190     // Optimistic Shashlyk
191     // 2%/E  + 8%/sqrtE + 2%
192     // A       B           C
193     Smear::Acceptance::Zone EmcalFwdInnerZone(ThetaFromEta ( 3.5 ),ThetaFromEta ( 2 )); // E
194     Smear::Device EmcalFwdInner(Smear::kE, "sqrt( pow ( 0.02,2 ) + pow( 0.8,2)*E + pow ( 0.02*E,2 ) )");
195     EmcalFwdInner.Accept.AddZone(EmcalFwdInnerZone);
196     EmcalFwdInner.Accept.SetGenre(Smear::kElectromagnetic);
197     det.AddDevice(EmcalFwdInner);
```

You may want to change:

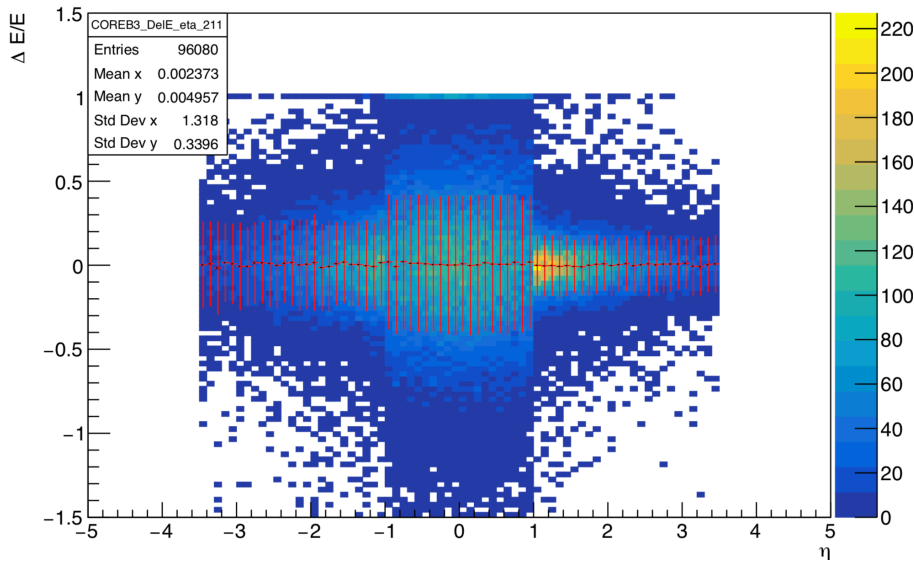
- Different Calo options
- Angular resolution (see below)

Calo Performance examples

ΔE / p electrons CORE



ΔE / p pions CORE



CORE: PID

```
117     // p < 10 GeV
118     Smear::Acceptance::Zone PidZone(ThetaFromEta(-4.0),ThetaFromEta(-4.0),
119                                     0., TMath::TwoPi(), // phi
120                                     0., TMath::Infinity(), // E
121                                     0., 10 ); // p
122     Smear::PerfectID Pid;
123     Pid.Accept.AddZone(PidZone);
124     Pid.Accept.SetCharge(Smear::kCharged);
125     Pid.Accept.SetGenre(Smear::kHadronic);
126     det.AddDevice( Pid );
```

For now, very simple PID! Just perfect π/K/p in p<10 GeV

You may want to change:

- Acceptance
- If you have a Matrix, you can use that
- Wait just a little for nσ support (see below)

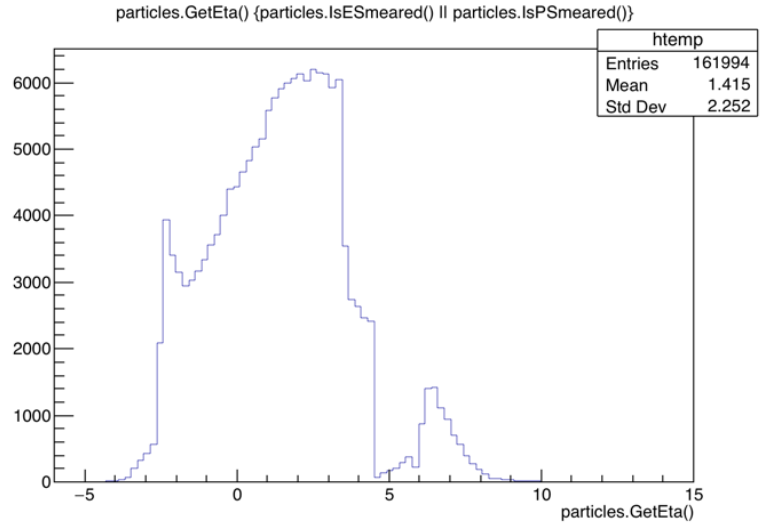
Far forward support?

- In **`SmearMatrixDetector_0_1_FF.cxx`**, added (rough) parameterization and acceptance from https://wiki.bnl.gov/eicug/index.php/Yellow_Report_Detector_Forward-IR
- You can copy and paste these lines
- Includes ZDC, B0, Off-momentum, Roman Pots

```
$ echo 'SmearTree(BuildByName("matrixff",275),
"ep_hiQ2.20x250.small.root")' | eic-smear

$  eic-smear ep_hiQ2.20x250.small.smear.root

eic-smear [] Smeared->Draw("particles.GetEta()",
"particles.IsESmeared() || particles.IsPSmeared()")
```



Future: Calorimetry Angular Resolutions

- Right now, it's assumed perfect – very unrealistic. Instead use:

Detector Matrix for the calorimeters

Source, slide 2

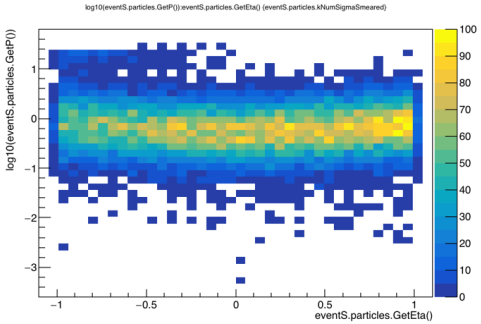
η	Nomenclature	EmCal						HCal			
		Energy resolution %	Spatial resolution mm	Granularity cm^2	Min photon energy MeV	PID e/π πsuppression	Technology examples*	Energy resolution %	Spatial resolution mm	Granularity cm^2	Technology solution
-3.5 : -2	backward	$2/\sqrt{E} \oplus 1$	$3/\sqrt{E} \oplus 1$	2x2	50	100	$PbWO_4$	$50/\sqrt{E} \oplus 10$	$50/\sqrt{E} \oplus 30$	10x10	Fe/Sc
-2 : -1	backward	$7/\sqrt{E} \oplus 1.5$	$3(6)/\sqrt{E} \oplus 1$	2.5x2.5 (4x4)	100	100	DSB:Ce glass; Shashlik; Lead glass	$50/\sqrt{E} \oplus 10$	$50/\sqrt{E} \oplus 30$	10x10	Fe/Sc
-1 : 1	barrel	$(10\text{-}12)/\sqrt{E} \oplus 2$	$3/\sqrt{E} \oplus 1$	2.5x2.5	100	100	W/ScFi	$100/\sqrt{E} \oplus 10$	$50/\sqrt{E} \oplus 30$	10x10	Fe/Sc
1 : 3.5	forward	$(10\text{-}12)/\sqrt{E} \oplus 2$	$3/\sqrt{E} \oplus 1$	2.5x2.5 (4x4)	100	100	W/ScFi Shashlyk, glass	$50/\sqrt{E} \oplus 10$	$50/\sqrt{E} \oplus 30$	10x10	Fe/Sc

Code ready for barrel and endcap, for any level of projectivity
Need R, z position and thickness, plus any changes from that table

Future: Better PID

NumSigmaPid class, based on https://gitlab.com/preghenella/pid

- Not suited (yet) for individual particles
- Not fully uniform
- Need some more agreement on interaction with momentum smearing



Relatively easy case-by-case
Will work with Pawel, others on DIRC and dRICH this week

Technical Points for Changes

- Copy the original script, change `BuildXX()` function name
- Add the name to `eicsmeardetectors.hh`

```
32   // experimental
33   Smear::Detector BuildCore_0_1_B3T();
34   Smear::Detector BuildCore_0_1( const double Bfield );
```

- Add to `CMakeLists.txt`

```
17 # Main target is the libeicsmear library
18 # staying away from wildcards and variables
19 add_library(
20    eicsmeardetectors
21    SHARED
22    ${CMAKE_CURRENT_BINARY_DIR}/eicsmeardetectorsDict.cxx
23    SmearMatrixDetector_0_2_B1_5T.cxx
24 [...]
25    SmearCore_0_1.cxx
26    )
```

- optional: Adapt `BuildByName.cxx` and `cint/LinkDef.h`

```
100     // -- d is the B field for tracking
101     if ( dname == "CORE_0_1"  ||
102          dname == "CORE" ) return BuildCore_0_1( d );
```

```
14   #pragma link C++ function BuildMatrixDetector_0_1;
15   #pragma link C++ function BuildMatrixDetector_0_2_B1_5T;
```

Summary

Often, I focus on running/smearing/analyzing MC output.

Today, I hope to convey:

- smearing scripts are long but **straightforward**
- Making changes to smearing scripts is **easy**
  - I put together CORE in a busy afternoon
  - Added arbitrary B field in about the same amount of time (mostly spent copy/pasting from a table and reading up on `std::valarray`)
- Big **improvements** to PID, angular resolutions coming soon – and your input is much appreciated!

Happy simulating, CORE!

BACKUP

Smearing

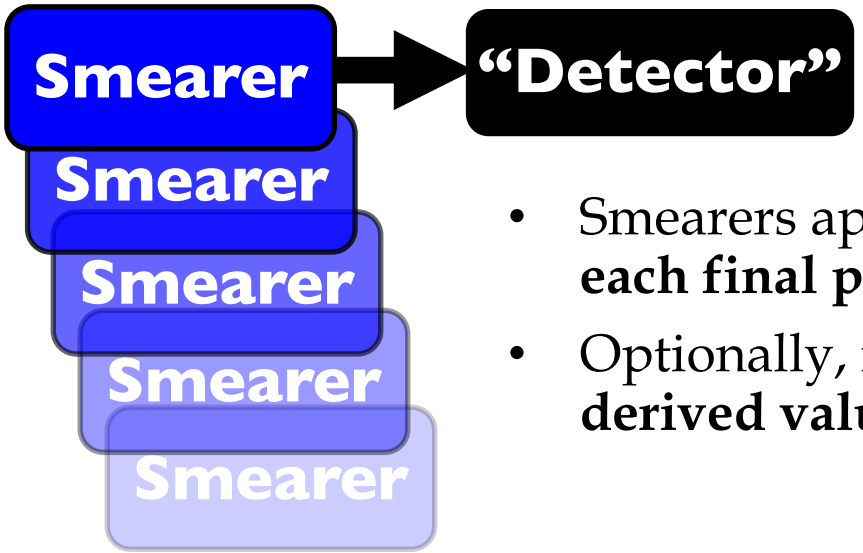
"**Smearer**" defines some element of performance + acceptance

- Standard Smearers are provided
- Define your own via inheritance

(single) quantity, X, to smear: E, p, θ, φ + Function defining σ(X) = f([E, p, θ, φ]) + Acceptance for X in E, p, θ, φ, pT, pZ

NOT a "physical detector"

Represents the **overall performance** in measuring a quantity.

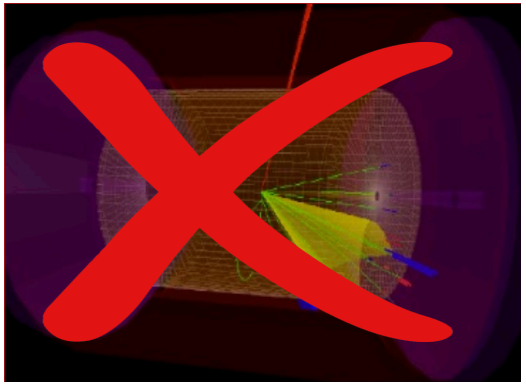


- Smearers applied to **each final particle**
- Optionally, recalculate **derived values** e. g. x, Q^2

Limitations

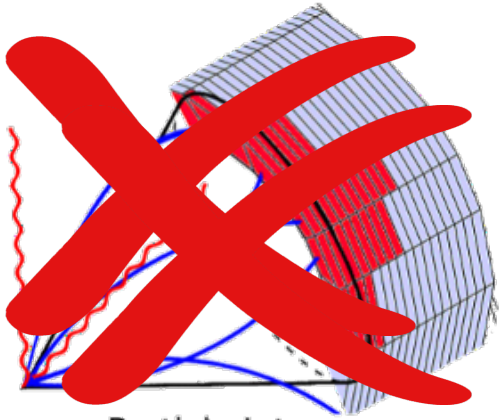
No concept of geometry, no B-Field

- no physically overlapping unrelated neutral and charged depositions
- Devices can and do have internal geometry though



No high-level analysis tools

- Fit into existing analysis chain
- Some logic exists for best kinematics calculation but (currently) no weighted mean for example
- But can DELPHES handle crossing angles etc.?



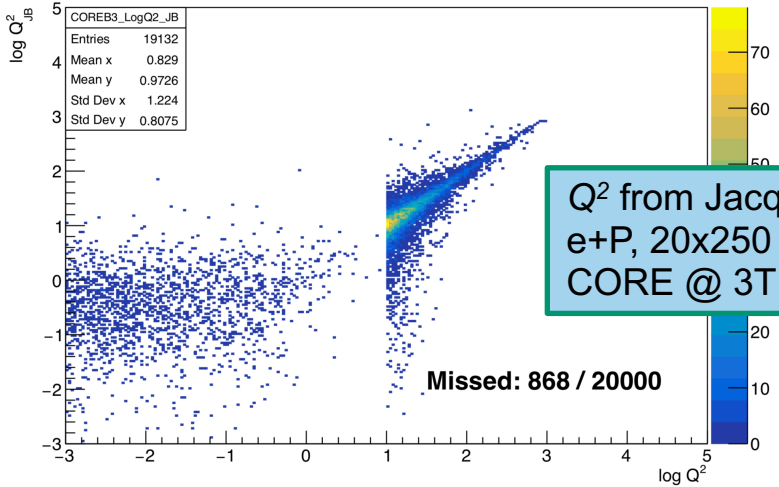
Vertex smearing untested

No decays, material budget, …

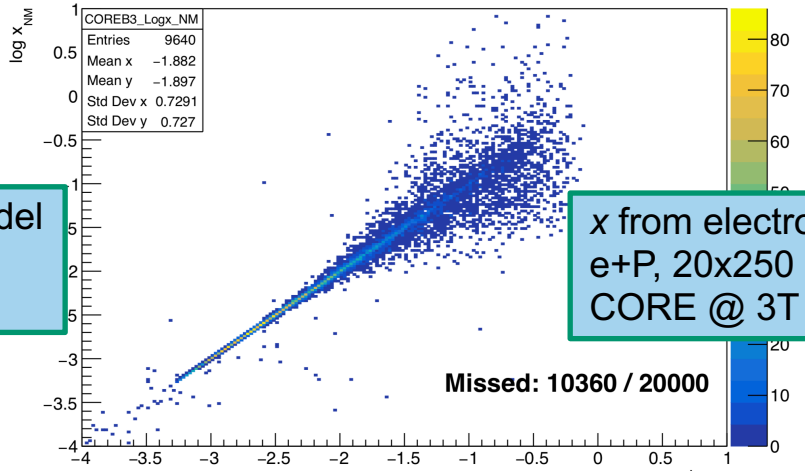
No efficiency → can be added

Min. E and p are supported but only at truth level

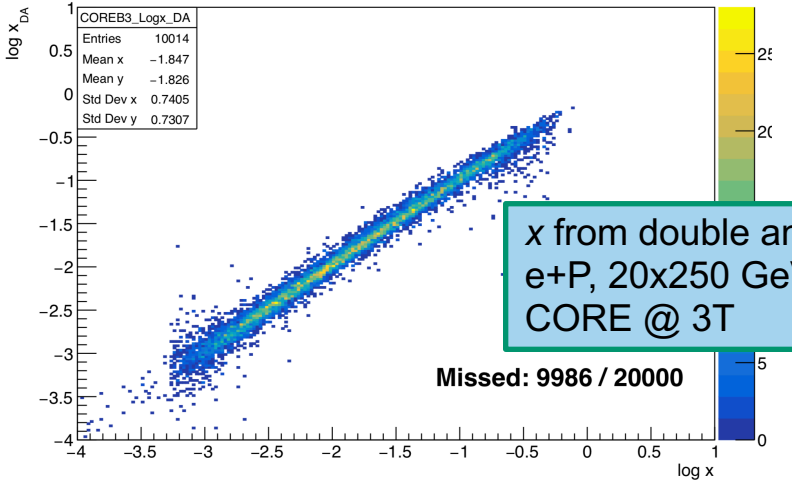
Derived Quantities



Q^2 from Jacquet-Blondel
e+P, 20x250 GeV
CORE @ 3T



x from electron
e+P, 20x250 GeV
CORE @ 3T



x from double angle
e+P, 20x250 GeV
CORE @ 3T

How to obtain

Necessary for installation:

- c++ and fortran compilers
- cmake 3.1+
- ROOT (6 is preferred)

Good to have:

- HepMC3, zlib

`gcc 4.8.5 is *just* enough for most things`

Even easier: singularity, fun4all, ESCalate, …
https://eic.github.io/
https://eic.github.io/software/eicsmear_generators_singularity.html

https://eic.github.io/software/eicsmear.html

Generators may need LHAPDF, CERNLIB, potentially FLUKA

Resources

- All hosted at github.com/eic (and https://gitlab.com/eic/mceg)
- Use issue tracker for bugs & requests!

- Slack channel: eicug.slack.com/#software-support
- Mailing list: eicug-software@eicug.org, eic-bnl-soft-l@lists.bnl.gov
- Contact: kkauder@bnl.gov

- Online users' guide: https://eic.github.io/software/eicsmear.html
- Class documentation:
  https://eic.github.io/doxygen/
  https://eic.github.io/doxygen/d9/dd8/namespaceSmear.html
  https://eic.github.io/doxygen/db/dfc/namespaceerhic.html