A root macro for tracking sim

F. Bedeschi
FCC-ee Physics Meeting
CERN, March 7, 2019

Outline

- The need
- The core
- Implementation
- Examples
- Fast simulation
- Examples
- Conclusions

The need

- IDEA tracking system still evolving
- Need fast turn around in evaluation of various options
  - Easy implementation of modified geometry
  - Easy change of detector performances
- Werner's formulas are useful but limited
  - Equal spacing
  - Uniform resolution
- Need realistic input for fast simulation
  - Full covariance matrix
  - Dependence on pt and polar angle

The core (1)

- Track fit χ^2 linearized in the fit parameters:

$$\chi^2 = \vec{d}^t S^{-1} \vec{d} \simeq (\vec{d}_0 - \vec{d}^* + \frac{\partial \vec{d}}{\partial \vec{p}} \cdot \Delta \vec{p})^t S^{-1} (\vec{d}_0 - \vec{d}^* + \frac{\partial \vec{d}}{\partial \vec{p}} \cdot \Delta \vec{p})$$

  - d/d* = predicted/measured distance of track from wire or pixel
  - p = track parameters
  - S = covariance of all measurements: resolution & MS
    - MS → worse resolution and non diagonal correlation terms

- Track parameter resolution depends on S and derivatives:

$$C^{-1} = \frac{1}{2} \frac{\partial^2 \chi^2}{\partial \vec{p} \partial \vec{p}} = A^t S^{-1} A, \text{ where } A = \frac{\partial \vec{d}}{\partial \vec{p}}$$

The core (2)

- Use full track helix for trajectories/acceptance, but
  - Keep only first outgoing branch
    - $\phi(R) = \phi_0 + \text{ArcSin}\{[RC+(1+CD)D/R]/(1+2CD)\}$
    - $z(R) = z_0 + \frac{\cot(\theta)}{C}\,\text{ArcSin}\left(C\sqrt{\frac{R^2-D^2}{1+2CD}}\right)$

- Some approximations in the calculation of derivatives;
  - CD<<1, |D| <<R
  - OK also for low momentum track
  - Full derivatives much more complex, but could be added later

Implementation (1)

- SolGeom class:
  - Fills geometry and draws it
    - Draws also the material
  - Each sub-detector can be turned on or off
  - All layers either:
    - Measurement or inert (for MS)
    - Measurement is axial (Rϕ), small angle stereo or 90 deg. (Rz)
    - Cylinder shell (const R) or disk (constant z)
  - Can write geometry to a text file
  - Can be initialized by reading text file
- First geometry implemented:
  - Same as full simulation for comparison – **NOT OPTIMIZED**

Implementation (2)

- Typical SolGeom geometry block

```
//
// Vertex  detector (inner)
if (fEnable[1])
{
        const Int_t NlVtx = 3;                          // Assume 3 vertex pixel layers
        Double_t rVtx[NlVtx] = { 1.7, 2.3, 3.1 };        // Vertex layer radii in cm
        Double_t lVtx[NlVtx] = { 11.0, 15.0, 20.0 };     // Vertex layer half length in cm
        for (Int_t i = 0; i < NlVtx; i++)
        {
                ftyLay[fNlay] = 1;                      // Layer type 1 = R (barrel) or 2 = z (forward/backward)
                fxMin[fNlay] = -lVtx[i] * 1.e-2;         // Minimum dimension z for barrel  or R for forward
                fxMax[fNlay] = lVtx[i] * 1.e-2;          // Maximum dimension z for barrel  or R for forward
                frPos[fNlay] = rVtx[i] * 1.e-2;          // R/z location of layer
                fthLay[fNlay] = 280.E-6;                // Thickness (meters)
                frlLay[fNlay] = 9.370e-2;               // Radiation length (meters)
                fnmLay[fNlay] = 2;                      // Number of measurements in layers (1D or 2D)
                fstLayU[fNlay] = 0;                     // Stereo angle (rad) - 0(pi/2) = axial(z) layer - Upper side
                fstLayL[fNlay] = TMath::Pi() / 2.;      // Stereo angle (rad) - 0(pi/2) = axial(z) layer - Lower side
                fsgLayU[fNlay] = 4.E-6;                 // Resolution Upper side (meters) - 0 = no measurement
                fsgLayL[fNlay] = 4.E-6;                 // Resolution Lower side (meters) - 0 = no measurement
                fflLay[fNlay] = kTRUE;                  // measurement flag = T, scattering only = F
                fNlay++; fBlay++;
                fNm++;
        }
}
```

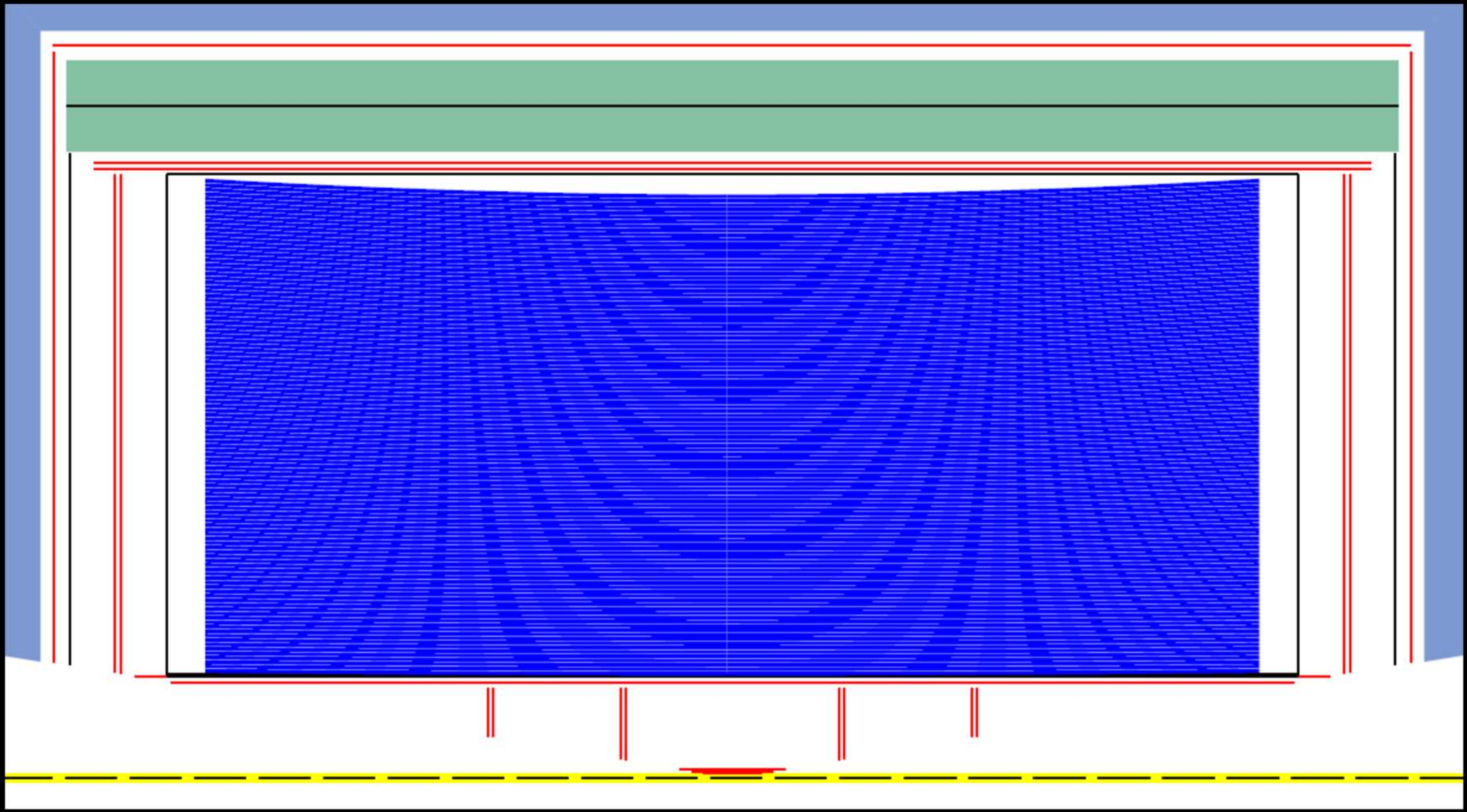
Examples (1)

- Initialize geometry and draw it

```
//
void SolGeoTest()
{
	//
	//	Init geometry
	//
	SolGeom *G;		// Init geometry
	const Int_t nDet = 9;
	Bool_t OK[nDet] = {	// Enable selected parts of the detector
		1,			// Beam pipe
		1,			// Inner VTX pixel layers
		1,			// Outer VTX layers
		1,			// Drift chamber
		1,			// Barrel Si wrapper
		1,			// Barrel pre-shower
		1,			// Forw. VTX pixel layers
		1,			// Forw. Si wrapper
		1 };			// Forw. pre-shower
	G = new SolGeom(OK);
	G->Draw();
```

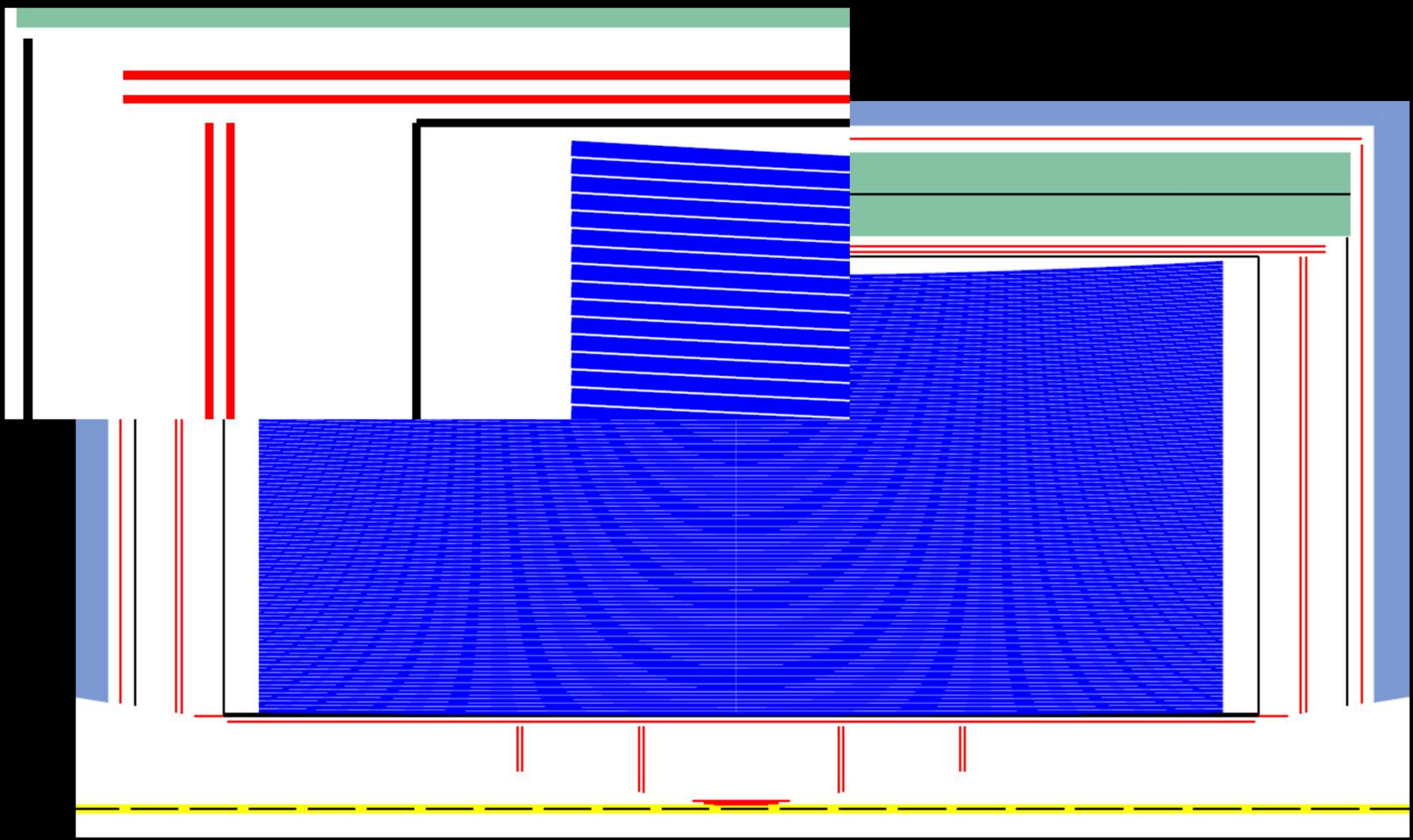
Examples (1)

❖ Initialize geometry and draw it



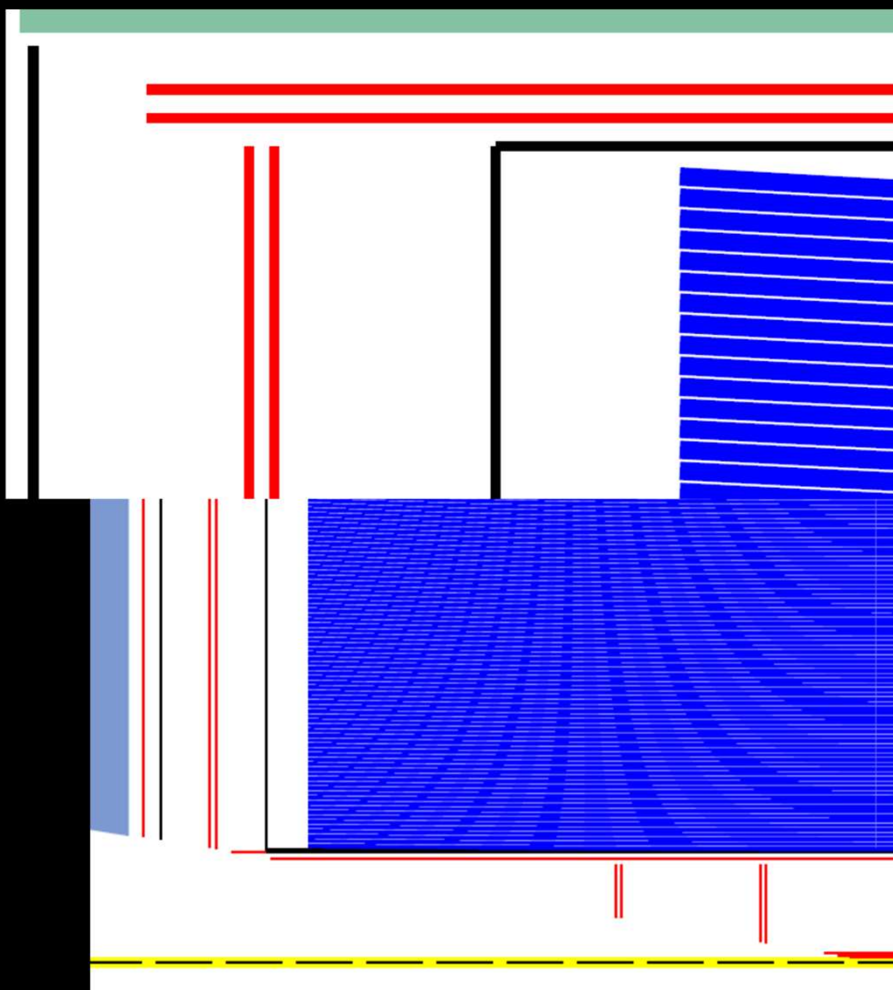
Examples (1)

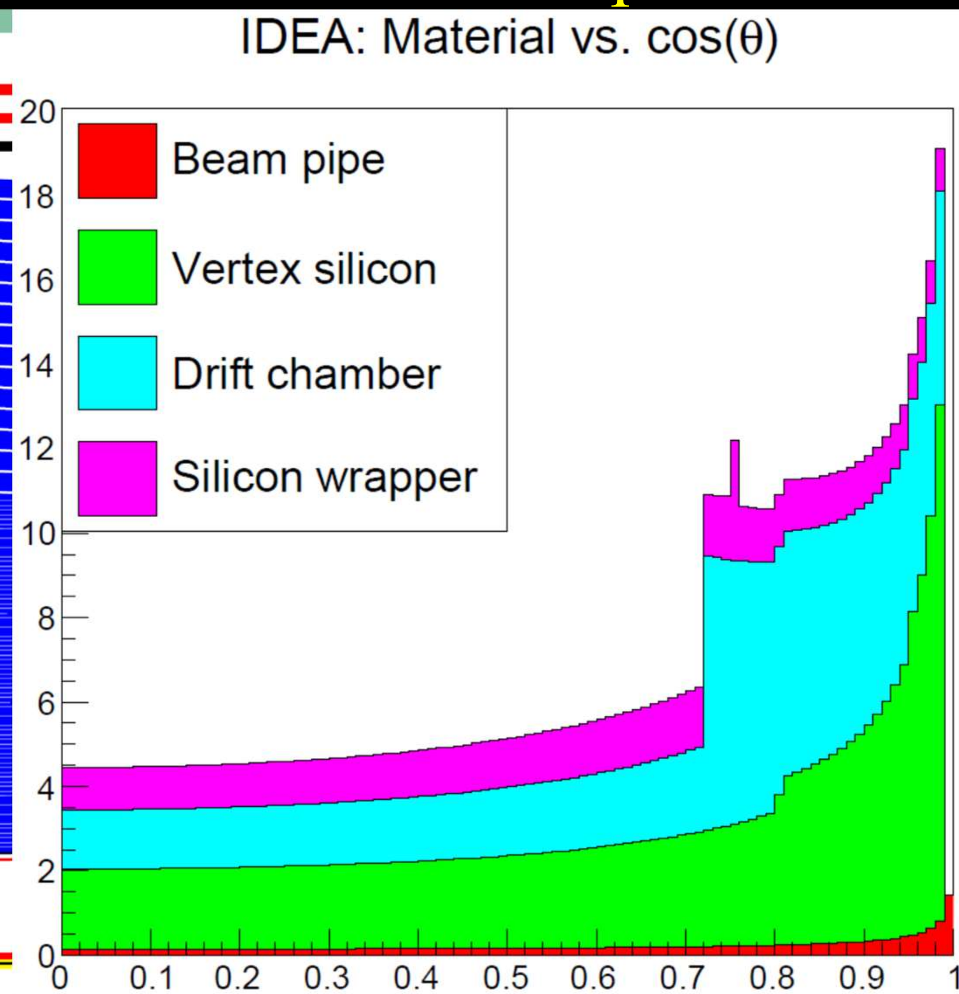
❖ Initialize geometry and draw it



Examples (1)

❖ Initialize geometry and draw it → material plot





Implementation (3)

- SolTrack class
  - Initialize with a geometry (SolGeom type) and two possible parameterizations (with automatic conversion):
    - $\vec{x}, \vec{p}$: postion and momentum vectors
    - C, ϕ_0, D, cot(θ), z_0: helix parameters
  - Finds intersection with any given layer (acceptance)
  - Calculates helix parameter covariance matrix
    - Include multiple scattering contributions with correlations

Examples (2)

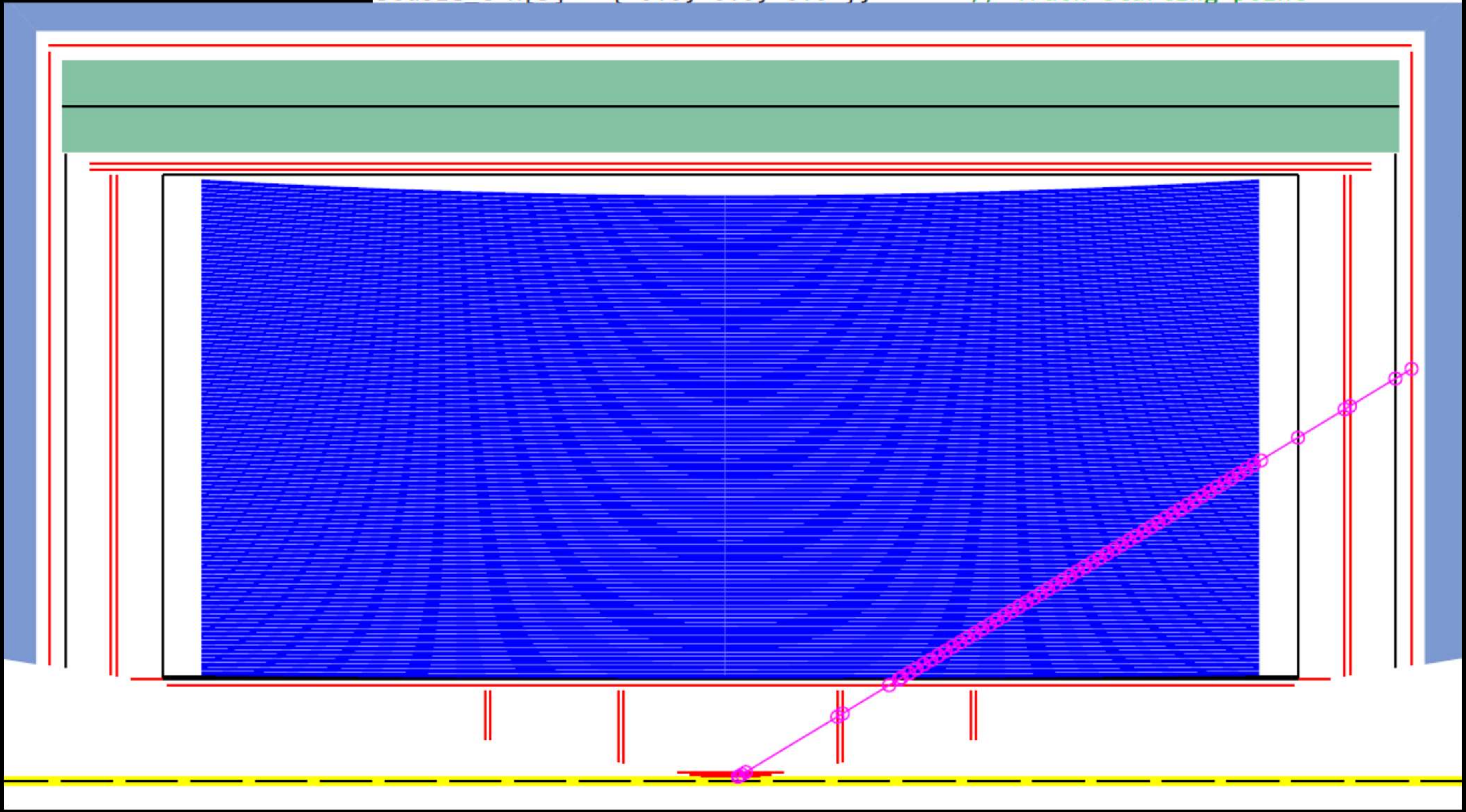
❖ Draw a track:

```
TCanvas *cc = G->cnv();                          // Get canvas with geo display
Double_t x[3] = { 0.0, 0.0, 0.0 };              // Track starting point
Double_t ppt = 1;                                // Track pt
Double_t ppz = ppt / TMath::Tan(th);             // Track pz
Double_t p[3] = { ppt, 0.0, ppz };              // Track momentum
SolTrack *trk = new SolTrack(x, p, G);           // Initialize track
TGraph *gr = trk->TrkPlot();                     // graph intersection with layers
gr->Draw("PLSAME");                              // plot track
```

Examples (2)

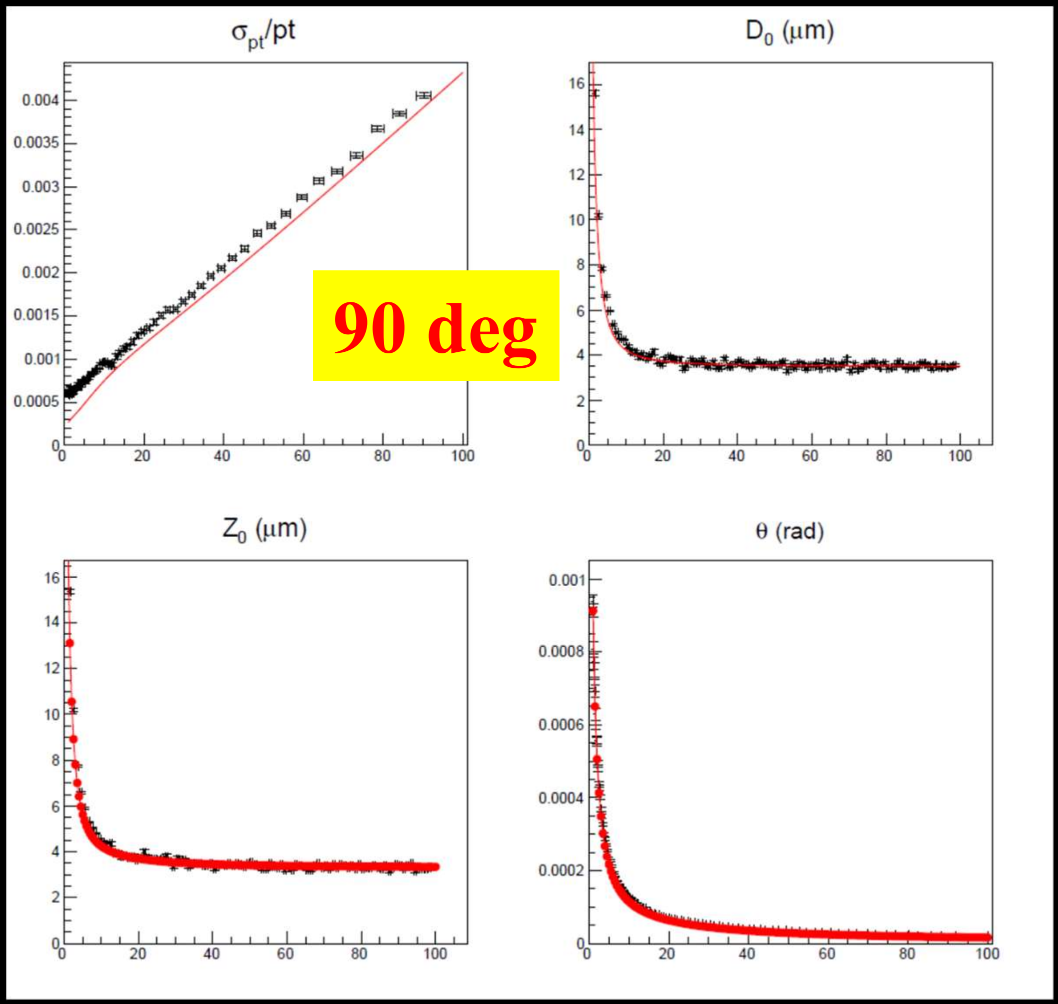
❖ Draw a track:

```
TCanvas *cc = G->cnv();                       // Get canvas with geo display
Double_t x[3] = { 0.0, 0.0, 0.0 };            // Track starting point
```



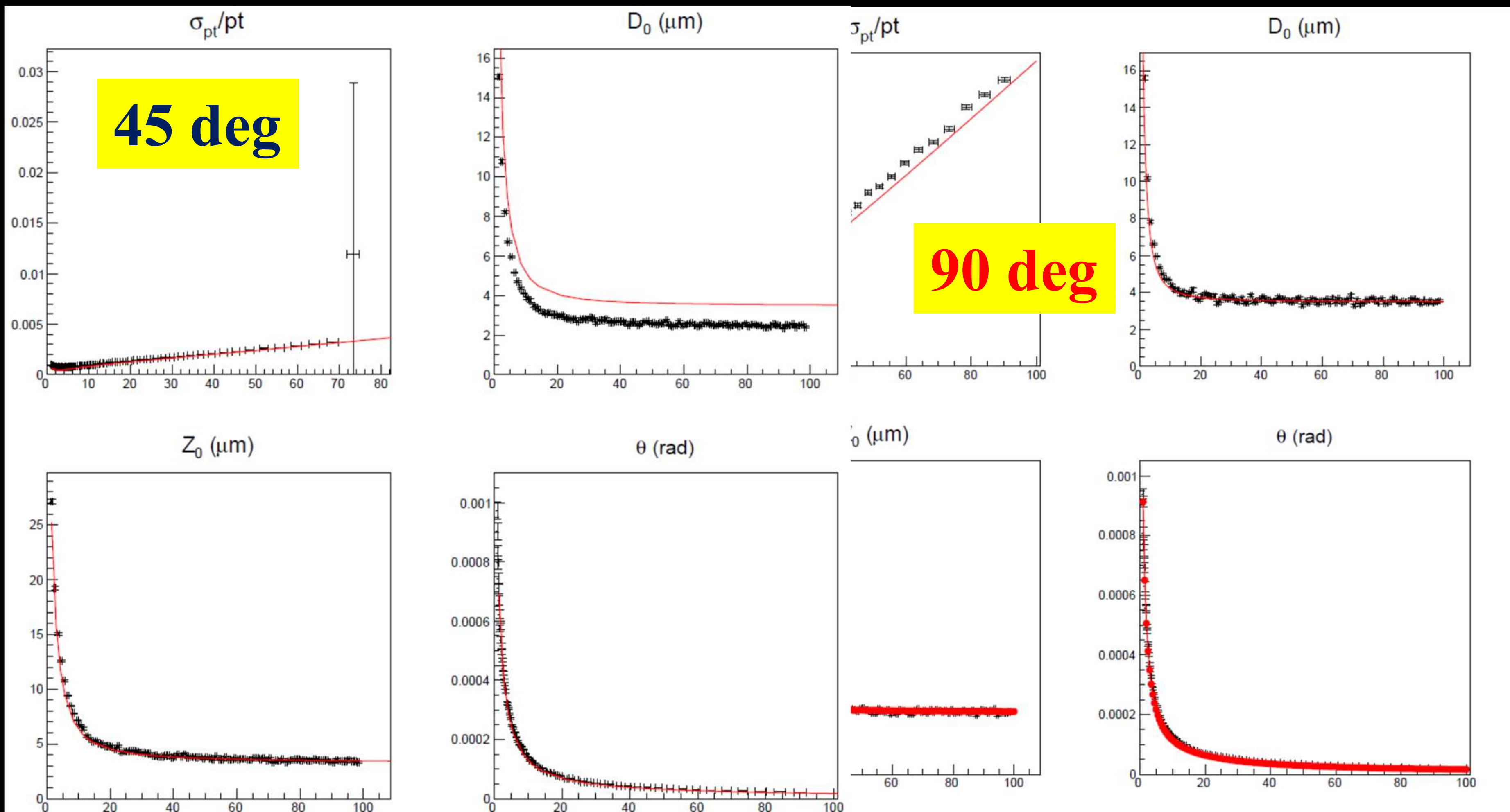
Examples (3)

❖ Compare resolution plots with full simulation (red = fast)



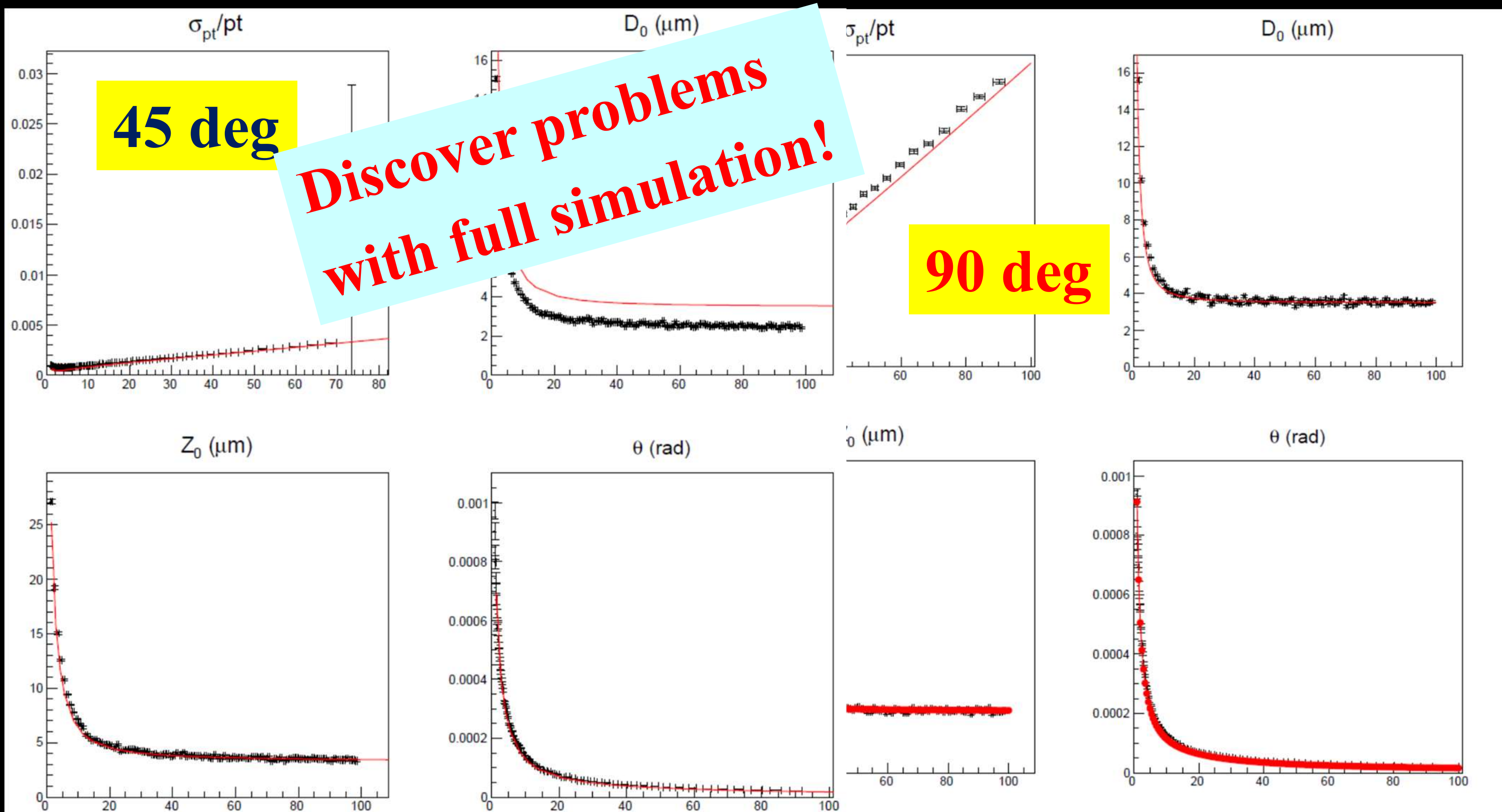
Examples (3)

❖ Compare resolution plots with full simulation (red = fast)



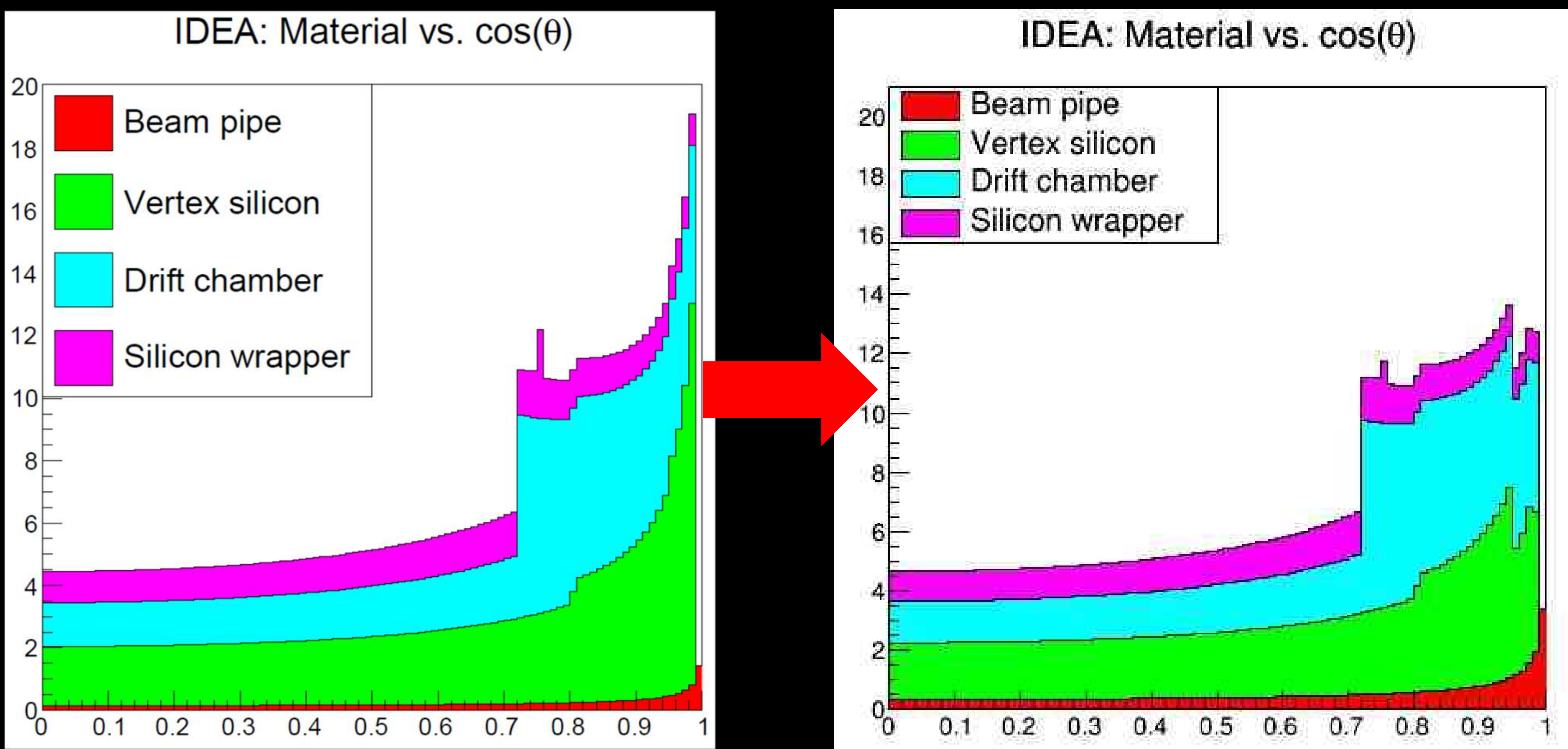
Examples (3)

❖ Compare resolution plots with full simulation (red = fast)



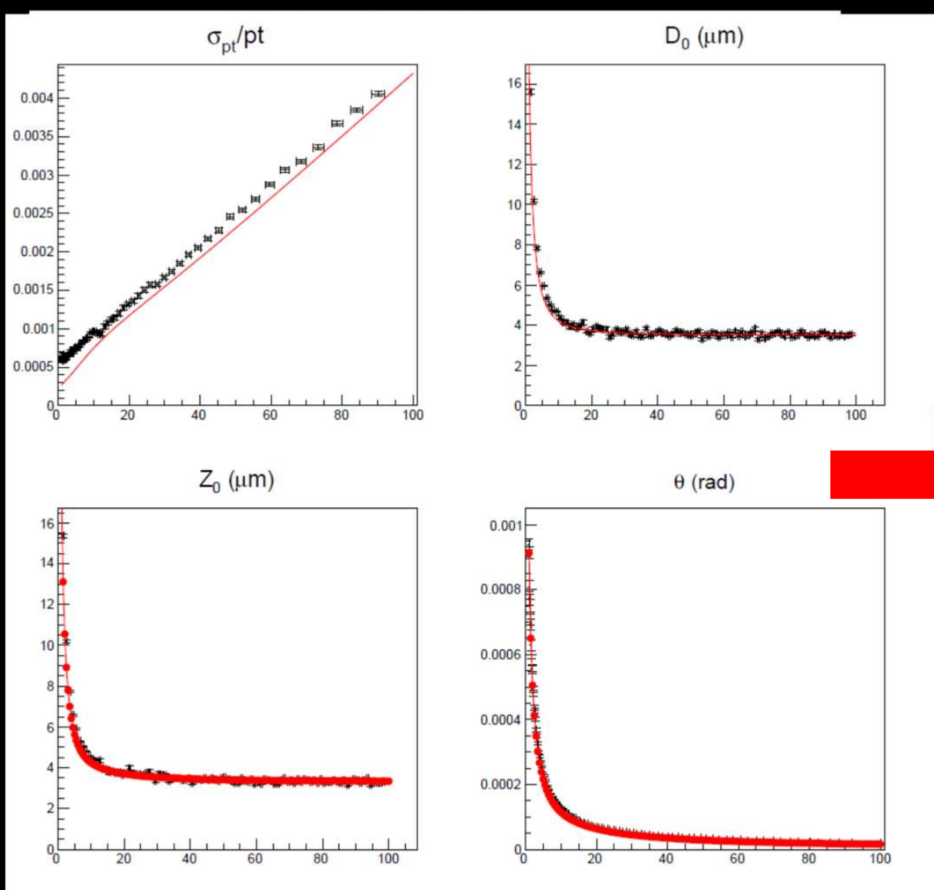
Examples (4)

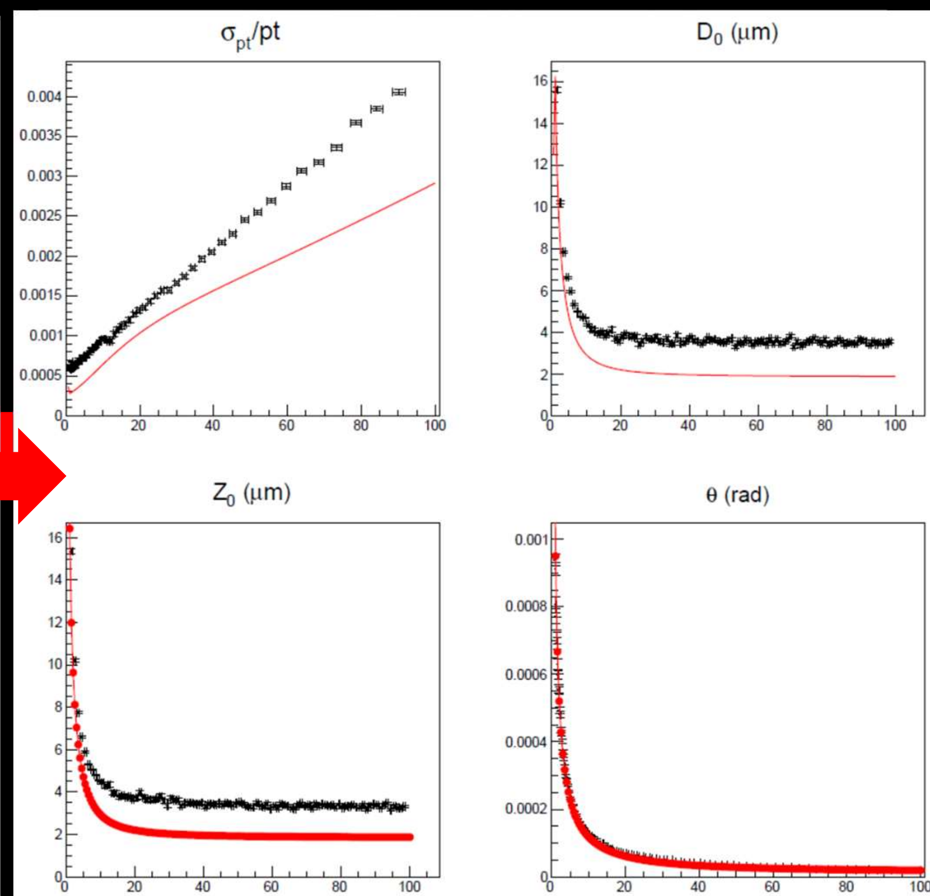
❖ Improve IDEA geometry



Examples (4)

- Improve IDEA geometry
- Use same resolutions as CLD for silicon





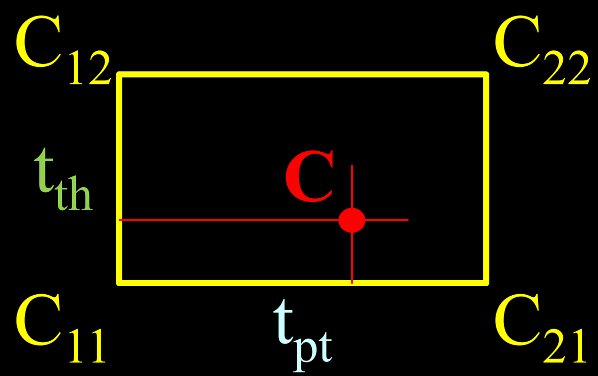
Make it faster

- SolTrack calculation of covariance matrix slow
  - Involves inversion of matrix ~ 120x120
- Solution:
  - Store pt-polar angle grid of matrices in .root file
  - Get any matrix by interpolation over 2D-grid
- Implementation with class SolGridCov

SolGridCov

❖ Methods:

- Creates TTree with covariance matrix branches
- Calculates covariance matrices at nodes with SolTrack
- Interpolates covariance matrix for any pt–polar angle
  - $C = C_{11}(1-t_{pt})(1-t_{th})+C_{12}(1-t_{pt})t_{th}$
    $+C_{21}t_{pt}(1-t_{th})+C_{22}\ t_{pt}\ t_{th}$
  - $t_{pt} = (pt-pt_{min})/(pt_{max}-pt_{min})$
  - $t_{th} = (th-th_{min})/(th_{max}-th_{min})$
  - Protect positive definitness

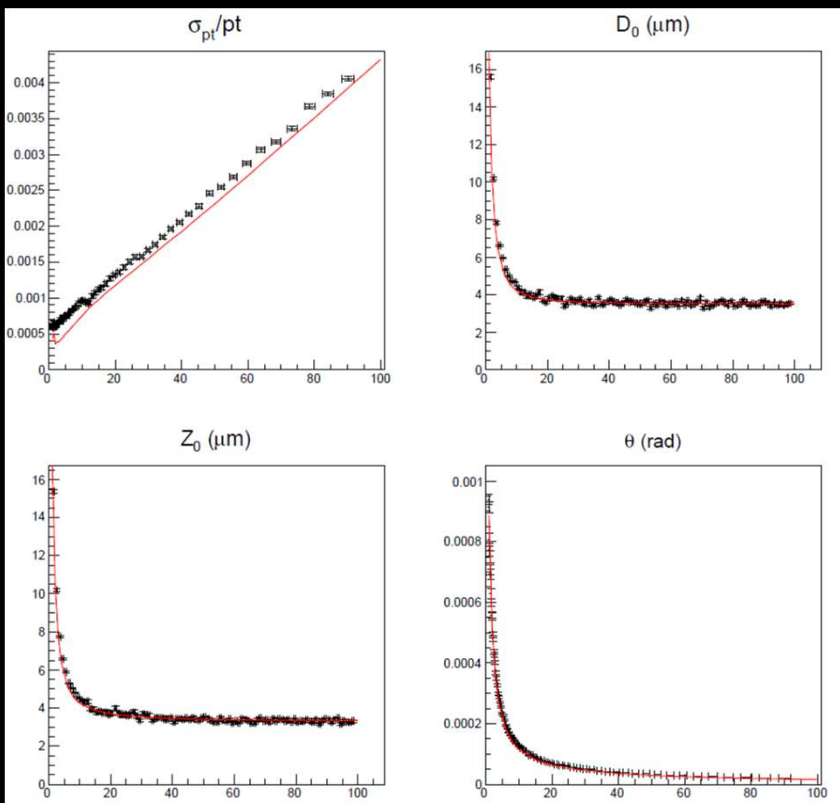


- Write/Read covariance matrix grid to file

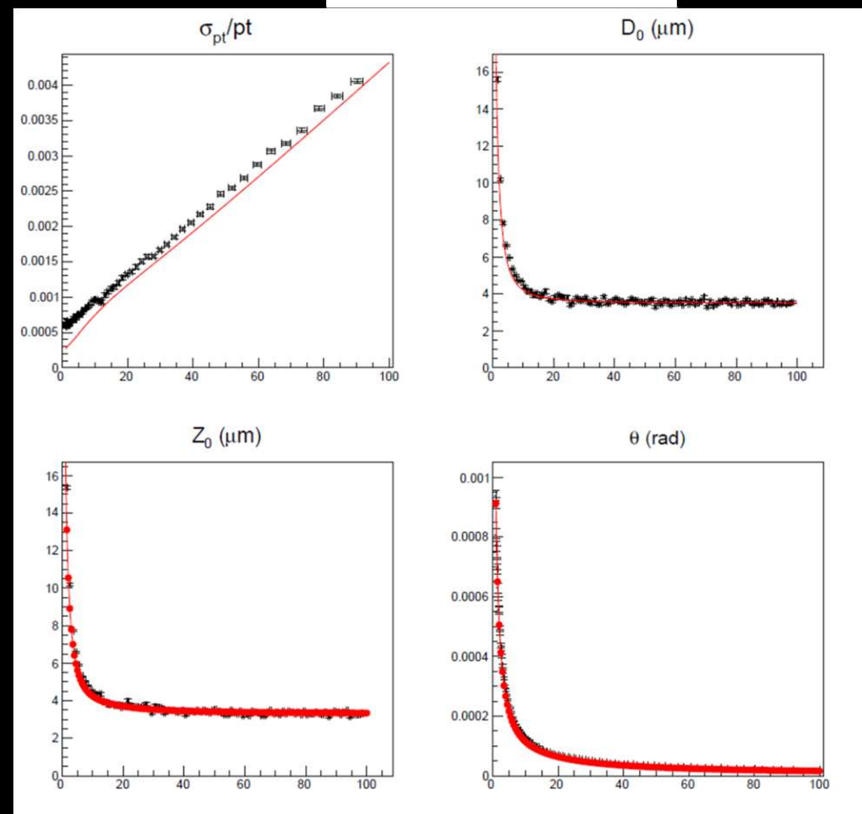
```
G = new SolGeom(OK);// Geometry with selected detectors
// Write covariance matrix grid to root file
SolGridCov *GC = new SolGridCov();
GC->Write("Cov.root", G);
```

Examples (5)

- Resolution comparison (90 degree)
  - This calculation: red line, Full simulation: black points



From grid



From SolTrack

Application to simulation

- Class ObsTrk:
  - Takes input perfect track → observed track
    - Conversion to helix parameters
    - Smear helix parameters according to appropriate covariance matrix
      - Use Choleski decomposition

Application to simulation

- Class ObsTrk:
  - Takes input perfect track → observed track
    - Conversion to helix parameters
    - Smear helix parameters according to appropriate covariance matrix
      - Use Choleski decomposition

$$
\begin{aligned}
C &= \text{Covariance matrix} \\
C &= U^T U \text{ (U is upper triangular matrix) - Choleski decomposition} \\
\vec{r} &= \text{vector of normal random numbers } \mu = 0,\ \sigma = 1 \\
\vec{x} &= U^T \vec{r} \to \vec{x} \text{ has covariance } C\text{. Proof:} \\
Cov(\vec{x}) &= < \vec{x} \cdot \vec{x}^T > = U^T < \vec{r} \cdot \vec{r}^T > U = U^T I U = U^T U = C
\end{aligned}
$$

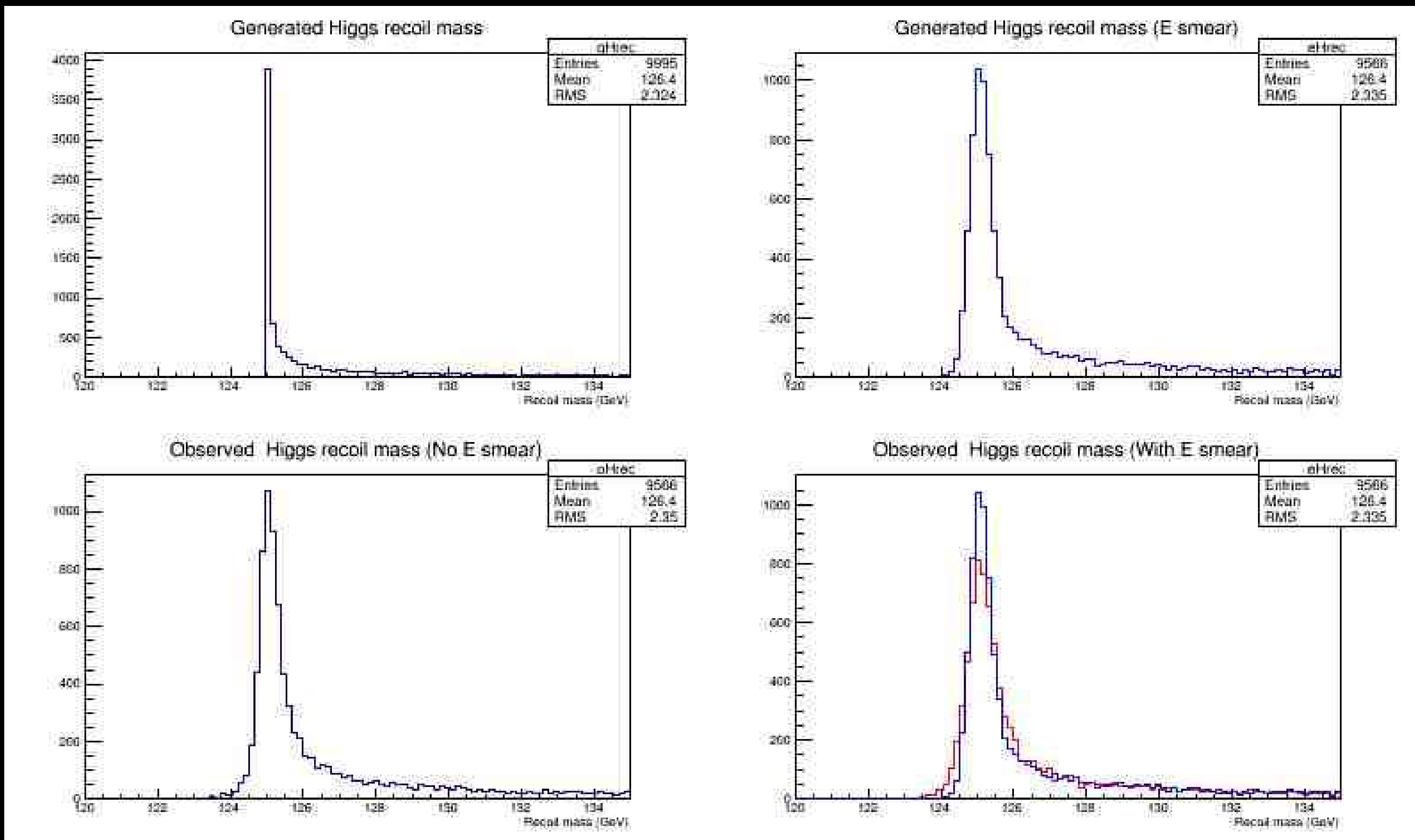
Example (6)

- Application in invariant mass calculation in Pythia generated events

```
// Initialize Geometry
SolGeom *G = new SolGeom();           // Initialize geometry
Double_t Bfield = G->B();             // Get B field in Tesla
// Initialize tracking resolution
SolGridCov *GC = new SolGridCov();
GC->Read("Cov.root");                         // Read in covariance array
.................................................................................
.................................................................................
//
// Now apply track resolution effects
//
  TVector3 tX(0.0, 0.0, 0.0);         // Set origin to (0,0,0)
  TVector3 tP1 = p1.Vect();           // Get generated momenta
  TVector3 tP2 = p2.Vect();
//
  ObsTrk *Tr1 = new ObsTrk(tX, tP1, Q1, Bfield, GC);  // Apply track resolution
  ObsTrk *Tr2 = new ObsTrk(tX, tP2, Q2, Bfield, GC);
  TVector3 obsP1 = Tr1->GetObsP();                    // Get smeared momenta
  TVector3 obsP2 = Tr2->GetObsP();
  Double_t E1 = TMath::Sqrt(M1*M1 + obsP1.Mag2());    // Smeared energies
  Double_t E2 = TMath::Sqrt(M2*M2 + obsP2.Mag2());
  TLorentzVector oP1(obsP1, E1);  // Fill smeared Lorentz vectors
  TLorentzVector oP2(obsP2, E2);
  TLorentzVector oPtot = oP1 + oP2;       // Total momentum 4-vector
  Float_t MiObs = oPtot.M();              // Observed invariant mass
```

Example (7)

❖ Higgs recoil from HZ ($Z \rightarrow \mu^+\mu^-$) – 0.1% CoM energy σ



Conclusion (1)

- Simple ROOT based classes to simulate tracking system resolution
  - Easy to change configuration
  - Perfect to compare options
- Used to feed Fast Sim a realistic covariance
  - Could be included inside DELPHES
- Validated on full simulation
- Demonstrated on specific physics process

Conclusions (2)

- Additional work:
  - Agree on baseline geometry and resolutions
  - Generate several geometry/covariance files for comparisons
    - CLD, CepC baseline, variations of IDEA baseline for optimization purposes
  - Use for physics studies involving tracking only
- Quite possible now for Oxford and FCC week
  - Some help would speed up things!
- All software described here can be found in
  - https://www.pi.infn.it/~bedeschi/RD_FA/Software/