



超级陶粲装置
Super Tau-Charm Facility


# FastSim package in STCF

**Xiaoshuai Qin (秦小帅)**

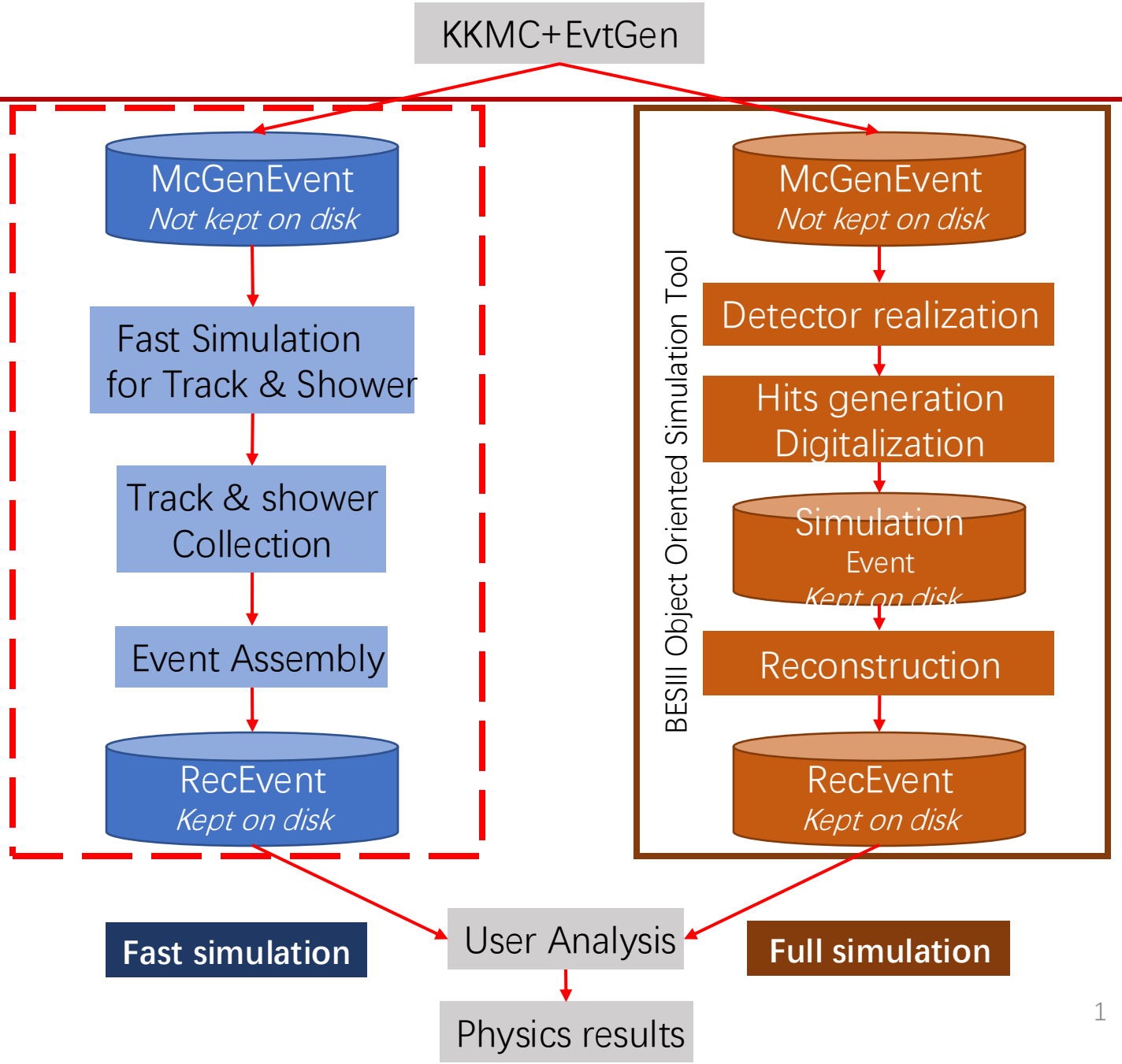
**Shandong University**

# 快速模拟工作原理

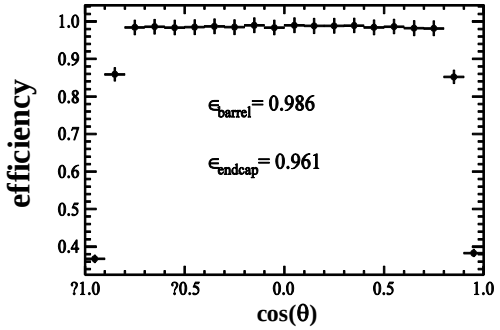
- 应对高亮度对撞实验中大统计量数据的产生、存储挑战

- 通过全重建样本得到探测器性能：
- 能动量分辨、位置分辨、重建效率、粒子鉴别效率等
- 细分到特定 P, cos(θ), ϕ 区间， 以便准确描述
- 获得探测器响应的CDF曲线、效率直方图

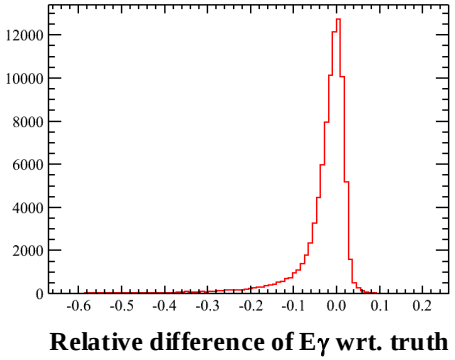
- 通过产生子拿到粒子四动量信息
- 利用探测器响应的CDF曲线、效率直方图进行抽样

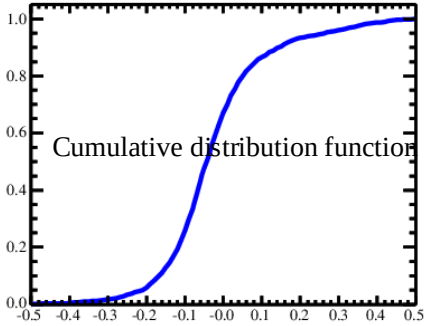


# 快速模拟的实现



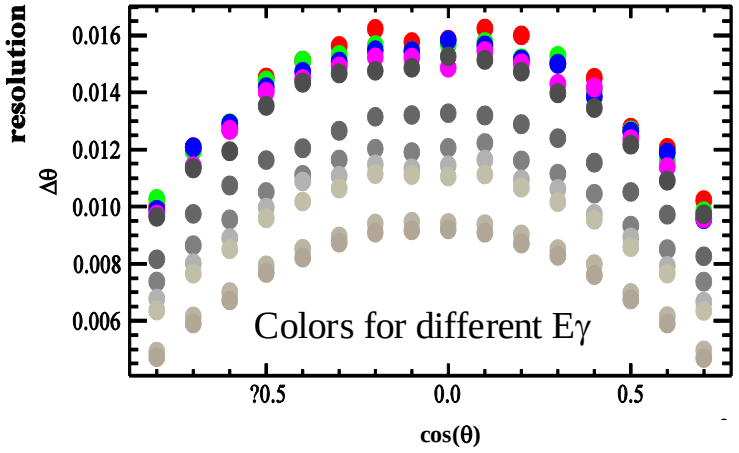

Efficiency sampling

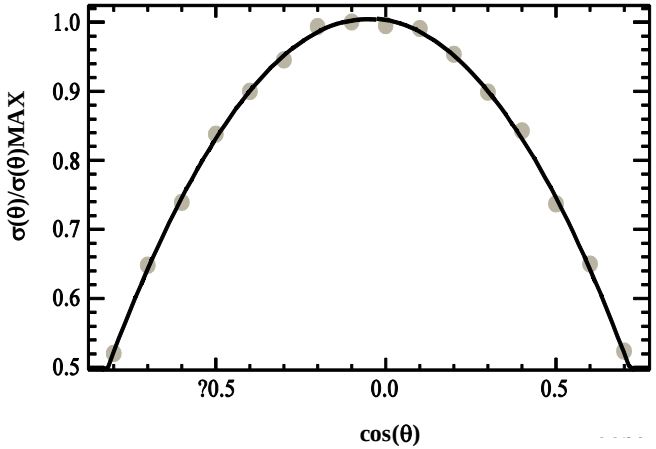



Smear

- Efficiency sampling
- Momentum smear
- Direction parameterization/smear



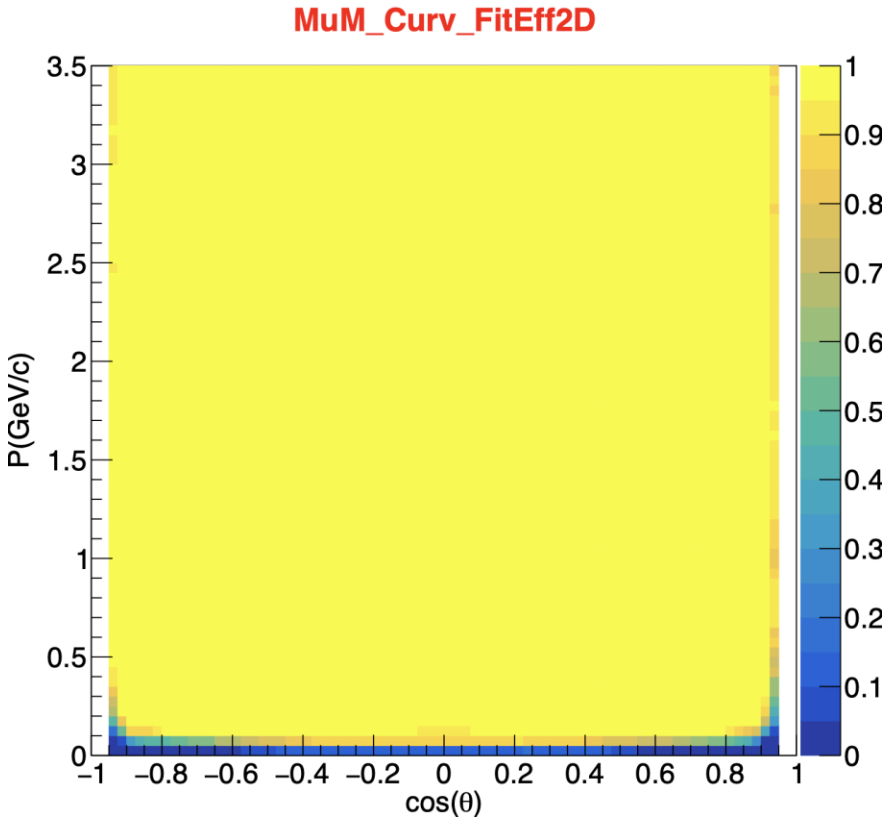




Parameterization
σ(pt, cos(θ) )

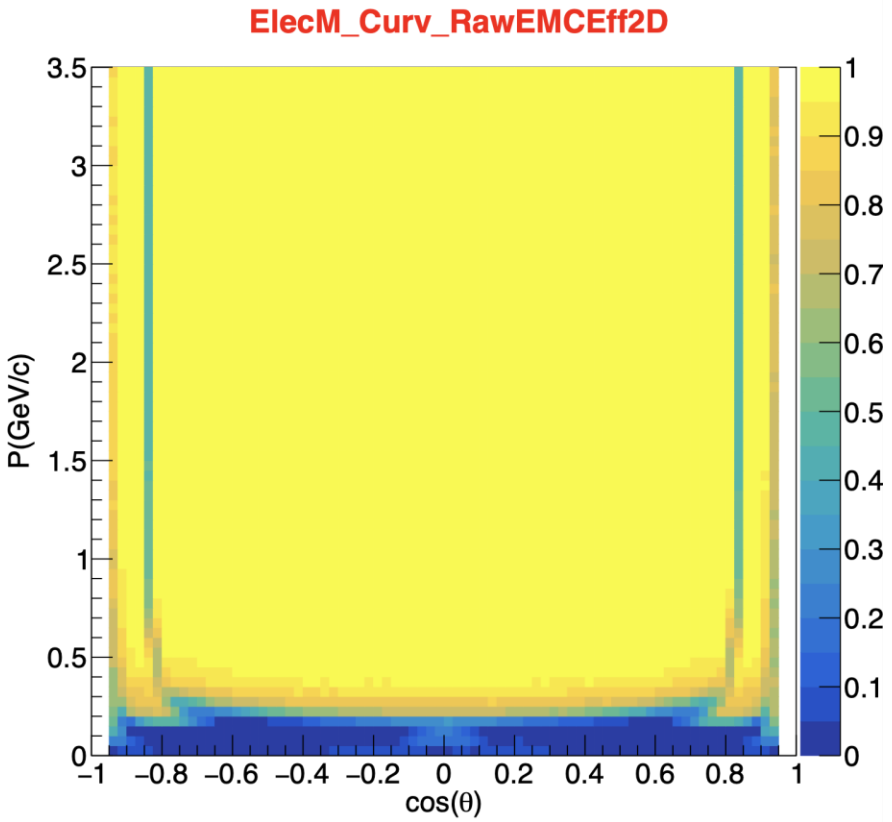
# OSCAR下重建效率与分辨

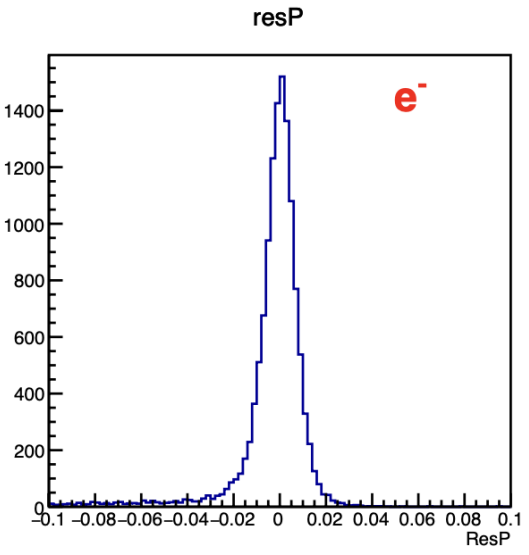
- 以原点产生的按角度、动量细分的单粒子重建样本为基础

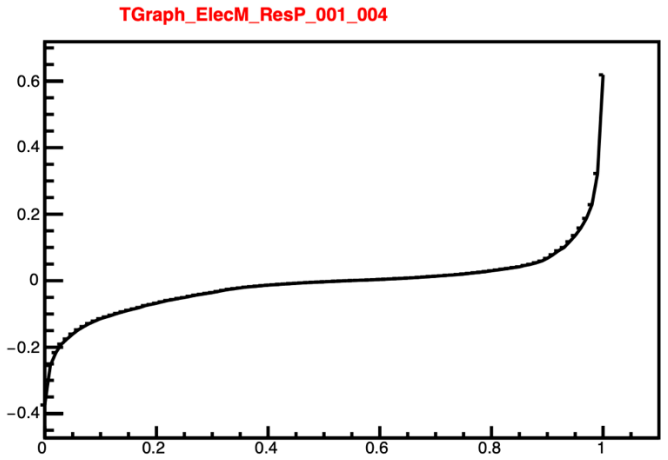
$\mu^-$径迹重建效率




$e^-$在ECAL中的重建效率



# 快速模拟算法简介

- **/lzufs/user/qinxs/tutorial/fastSim/algs**

OSCAR 2.6.0未包含最近的更新，
gitlab上获取最新的算法包
或近期使用可到这个目录下拷贝

- **算法包：$OFFLINE/Simulation/FastSim/**
- **两个算法类:**
  - **FastTruthWriter**:
    通过产生子给出的信息构建MCParticleCol
  - **FastSim**:
    快速模拟的主算法

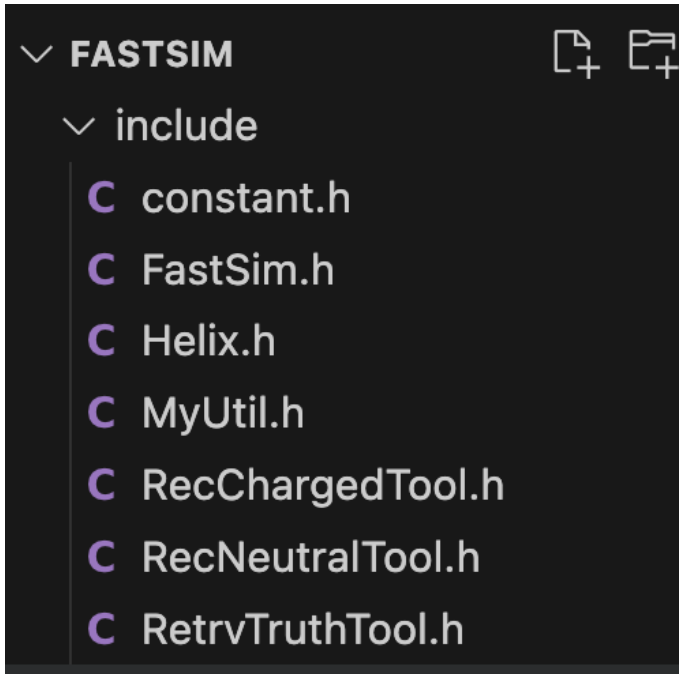
```
CMakeLists.txt
FastSim
FastValidAlg
GlobalPID
README
```

- Analysis/FastValidAlg 简单的辅助算法
  快速查看快模拟实现了哪些类的具体数据

```
import FastValidAlg
alg = task.createAlg("FastValidAlg")
alg.property("check").set(True)
alg.setLogLevel(4)
```

# FastSim 快速模拟主算法

- FastSim算法类
- RetrvTruthTool: 读取MCParticleCol，并构建带电粒子、中性粒子的truth信息(结构体)
- 分别调用RecChargedTool, RecNeutralTool完成效率、分辨的抽样(快速模拟)
- FastSim::evtAssemble() 组装成ReconstructedParticle




```
class FastSim: public AlgBase{
  FastSim(const std::string& name);
  ~FastSim();
  bool initialize();
  bool execute();
  bool finalize();
  bool copyMCParticles();
private:
  void debug( bool tmp=false ) { m_debugSim=tmp; }
  void clearVector();
  bool clearTDSCollection();
  bool regMdcTracks();
  bool regECALShowers();
  bool regMUDTracks();
  bool evtAssemble();
  void setCanTrack( MutableCanTrack&, recchg& );
  void setTrackerRecTrack( MutableTrackerRecTrack&, recchg& )
  void setRecECALShower( MutableRecECALShower&, recgam& );
  void setRecChgECALShower( MutableRecECALShower&, recchg& );
  void setRecChgMUDTrack( MutableMUDTrack&, recchg& );
```

# FastSim 快速模拟主算法

- RecChargedTool: 带电径迹重建 $e, \mu, \pi, K, p$
- RecNeutralTool: 中性粒子重建 $\gamma$ , $K_L$, Nbar

```
class RecNeutralTool : public ToolBase{
    void setRunEvtNum( int run, int evt
    void setResScales( double ene=1.0, d
    { m_scaleResEne=ene; m_scaleResThe=t

    /// do the neutral reconstruction;
    bool recGamEmcShower( trutrk&, recga
    int        effGamEmcShower( double,
    //bool resGamEmcShower( std::string,
    bool resGamEmcShower( int, double, i
    //bool resGamEmcShower( std::string,

    /// get the function;
    double     gamEneError( double, doub
```

```
class RecChargedTool: public ToolBase{
    /// do the charged reconstruction;
    bool recChargedTrk( trutrk, recchg&
    /// int    enebinGamShower( HepLore
    /// int    catgryGamShower( HepLore
    int  effChgMDCTrack( double, double
    int  PIDChgTrack( double, double, d
    int  STCFPIDChgTrack( double, doubl
    int  PIDMucChgTrack( double, double
    int  PIDSTCFMucChgTrack( double, do
    double EMCChgTrack( double, double,
    bool RecHelix(HepPoint3D, double, d
    bool resChgTrack( std::string, doub
    bool resChgTrack( std::string, doub
    bool HelixErrChgTrack( double, std:
```

# FastSim 快速模拟主算法

- FastSim算法：
  - 实现模拟的能动量范围：0.03-3.5GeV
  - $e, \mu, \pi, K, p, \gamma$ , $K_L$, Nbar

- 重建性能的快速设置，便于探测器设计的预研究 （分析用户大多数时候不需要关心）

```
declProp( "gamEneScale" , par_gamEneScale=1.0 );
declProp( "gamTheScale" , par_gamTheScale=1.0 );
declProp( "gamPhiScale" , par_gamPhiScale=1.0 );
declProp( "gamRecEffScale" , par_gamRecEffScale=1.0 );
declProp( "chgSigXY"            , par_chgSigXY=-1 );   // sigma_xy from MDC, unit: mum
declProp( "chgSigZ"             , par_chgSigZ=-1 );     // sigma_z from MDC, unit: mum
declProp( "chgRecEffScale" , par_chgRecEffScale=1.0 );
declProp( "chgVrScale"  , par_chgVrScale=1.0 );
declProp( "chgVzScale"  , par_chgVzScale=1.0 );
declProp( "vtxRadScale" , par_vtxRadScale=1.0 );
declProp( "vtxZdrScale" , par_vtxZdrScale=1.0 );
```

# FastValidAlg 简单的辅助算法

- 利用输出算符重载快速查看数据信息
  - std::ostream& operator<<(std::ostream& o, const ReconstructedParticle& value)

```
for (size_t i=0; i<m_recParticle->size(); ++i) {
  auto recpar = m_recParticle->at(i);
  auto mcpar =recpar.getMCParticle();
  auto momentum = recpar.getMomentum();
  cout<<"preMc,"<<count[0]<<","<<momentum<<"," \
    <<i<<","<<m_recParticle->size()<<endl;
  if(!mcpar.isAvailable()) continue;
  cout<<"RecPar[----------------"<<endl;
  cout<<recpar<<endl;
  cout<<"RecPar----------------]"<<endl;
  validMC=1;
  auto track = recpar.getTrack();
  auto shower = recpar.getShower();
  if(track.isAvailable()) {
    cout<<"TrackerRecTrack[-------"<<endl;
    cout<<track<<endl;
    cout<<"TrackerRecTrack-------]"<<endl;
    auto hypo = track.getTrackHypos(2);
    if(!hypo.isAvailable()) continue;
    cout<<"TrackerHypoTrack[------"<<endl;
    cout<<hypo<<endl;
    cout<<"TrackerHypoTrack------]"<<endl;
  }
```

全重建样本

```
RecPar[----------------
 id: 300000000
 trackID : 0
 energy : 0
 momentum : 0.753494 1.10499 0.583928
 referencePoint : 0 0 0
 charge : 1
 mass : 0
 MCParticle : 20000012
 RICHPid : 230000000
 track : 110000000
 dEdX : 140000000
 shower : 180000004
 mudTrack : 270000000
 mudCluster : 4274967294
```

快模拟样本

```
RecPar[----------------
 id: 90000000
 trackID : 0
 energy : 0
 momentum : 0.0756972 -0.0368283 -0.186064
 referencePoint : 10000 10000 10000
 charge : 1
 mass : 0
 MCParticle : 20000008
 RICHPid : 4274967294
 track : 40000000
 dEdX : 4274967294
 shower : 70000000
 mudTrack : 60000000
 mudCluster : 4274967294
```

# 快速模拟样本产生

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-

import Sniper

task = Sniper.Task("task")
task.setLogLevel(4) #higher is less

import os
topdir = os.getenv('OFFLINETOP')

import StcfRndmGenSvc
sSvc=task.createSvc("StcfRndmGenSvc/hSvc")
sSvc.property("RndmSeed").set(28133)
```

```
#****************** KKMC
import KKMC
detalg0 = task.createAlg("KKMC")
detalg0.property("CMSEnergy").set(3.097)
detalg0.property("BeamEnergySpread").set(0.0011)
detalg0.property("NumberOfEventPrinted").set(0)

detalg0.property("GenerateResonance").set(True)
detalg0.property("GenerateContinuum").set(False)
detalg0.property("GenerateDownQuark").set(False)
detalg0.property("GenerateUpQuark").set(False)
detalg0.property("GenerateStrangeQuark").set(False)
detalg0.property("GenerateCharmQuark").set(True)
detalg0.property("GenerateMuonPair").set(False)
detalg0.property("GenerateTauPair").set(False)
detalg0.property("GenerateRho").set(False)
detalg0.property("GenerateOmega").set(False)
detalg0.property("GeneratePhi").set(False)
detalg0.property("GenerateJPsi").set(True) #3096
detalg0.property("GeneratePsiPrime").set(False) #3686
detalg0.property("GeneratePsi3770").set(False)
detalg0.property("GeneratePsi4030").set(False)
detalg0.property("GeneratePsi4160").set(False)
detalg0.property("GeneratePsi4415").set(False)
detalg0.property("GeneratePsi4260").set(False)
detalg0.property("ParticleDecayThroughEvtGen").set(True)
detalg0.property("RadiationCorrection").set(False)
#****************** KKMC

#****************** EvtGen
import StcfEvtGen
detalg = task.createAlg("EvtDecay")
detalg.property("userDecayTableName").set("rhopi.dec") # replace DecayDecDir
#****************** EvtGen
```

# 快速模拟样本产生

```
#******************** EvtGen
import StcfEvtGen
detalg = task.createAlg("EvtDecay")
detalg.property("userDecayTableName").set("rhopi.dec") # replace DecayDecDir
#******************** EvtGen

#################### generator setting end

import PodioDataSvc
dsvc = task.createSvc("PodioDataSvc")

import PodioSvc
oSvc = task.createSvc("PodioOutputSvc/OutputSvc")
oSvc.property("OutputFile").set("fastsim.rhopi.root")

#******************** FastSimulation
import FastSim
fasttw= task.createAlg("FastTruthWriter")
#fasttw.property("Boost").set(False)  // switch off boost if you need
fastsim = task.createAlg("FastSim")

#******************** FastSimulation

task.setEvtMax(10000)
task.show()
task.run()
```

产生子设置

快速模拟算法

# 运行速度和存储空间

| Single Track | Fast Sim | 1000万样本 |
|---|---|---|
| CPU time | <1ms / event | ~2hour |
| Event size | ~ 1Mb/k | ~10G |
| $\Lambda\bar{\Lambda}$(4 tracks) | Fast Sim | 1000万样本 |
| CPU time | ~2ms / event | ~5hour |
| Event size | ~ 2Mb/k | ~20G |

- 物理过程统计量特别大，可借助快速模拟开展预研究

# 与全模拟的差别

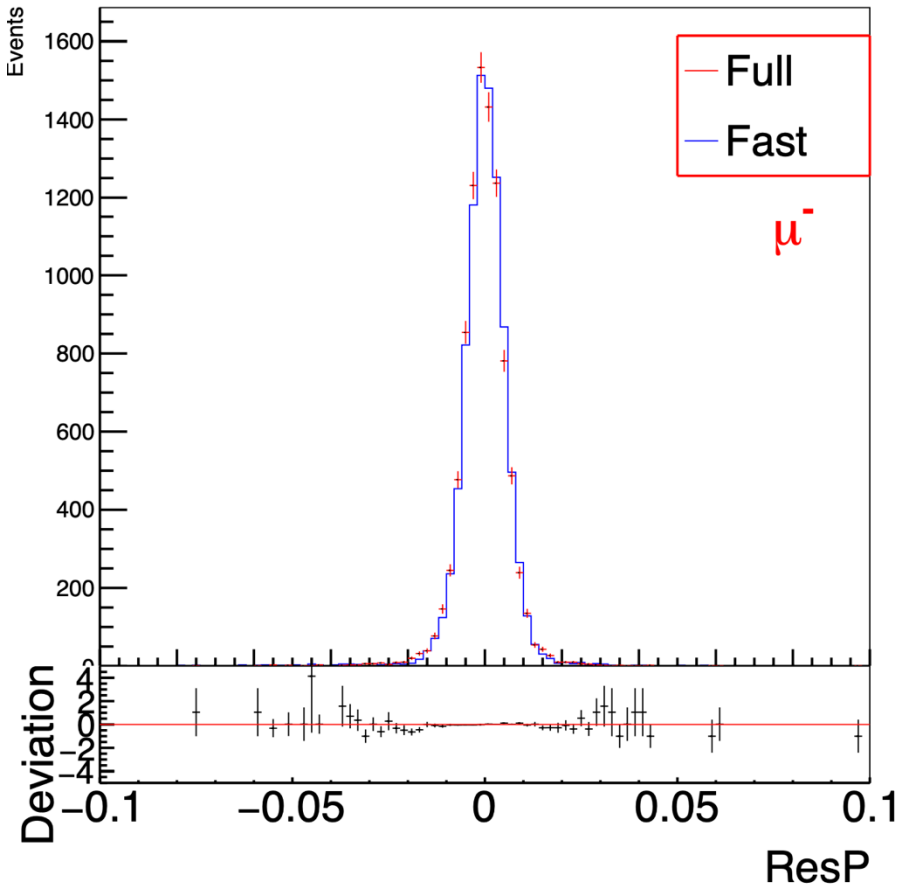
- 没有探测器几何，没有Geant4实现粒子输运
  - 没有击中信息
  - 没有衰变信息?
  - 没有电子对产生（Gamma Conversion）
  - 没有ExtTrack，没有径迹的外推信息
    - 中性径迹已经关联好
- 没有子探测器的测量量，比如dE/dx, RICH likelihood, MUD likelihood (有ECAL沉积能量)
- PID: 有PID的判选结果，对应的有虚构的概率, 似然值

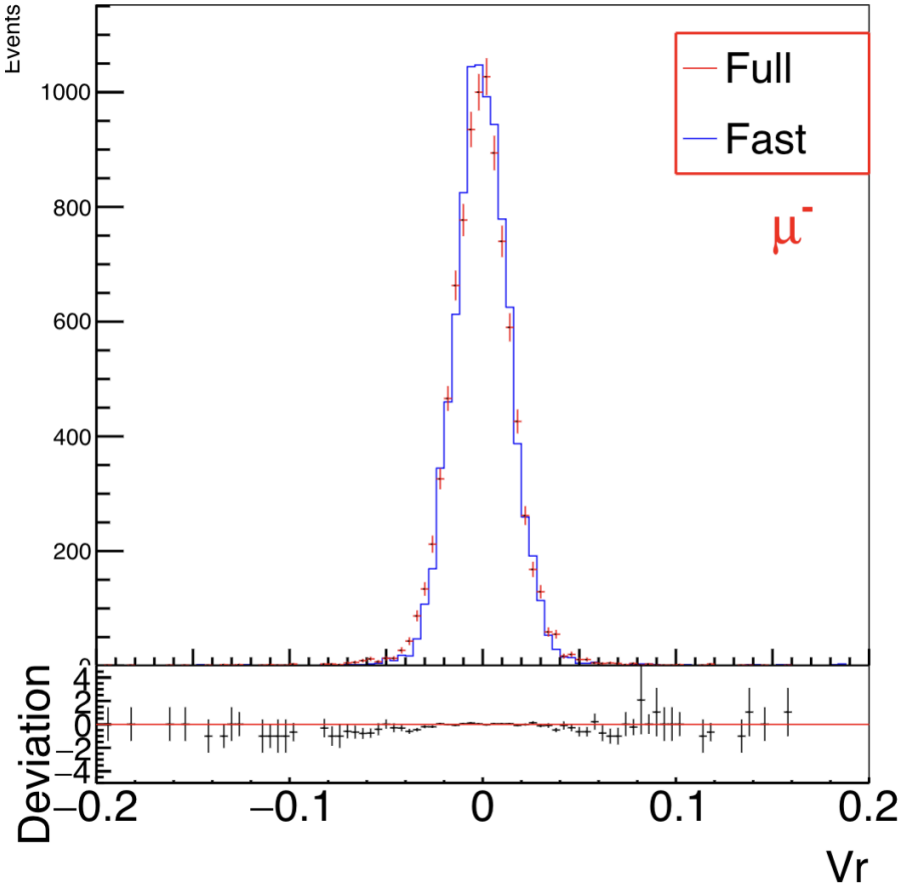
# 分析算法中要注意的地方

- 没有HitCollection

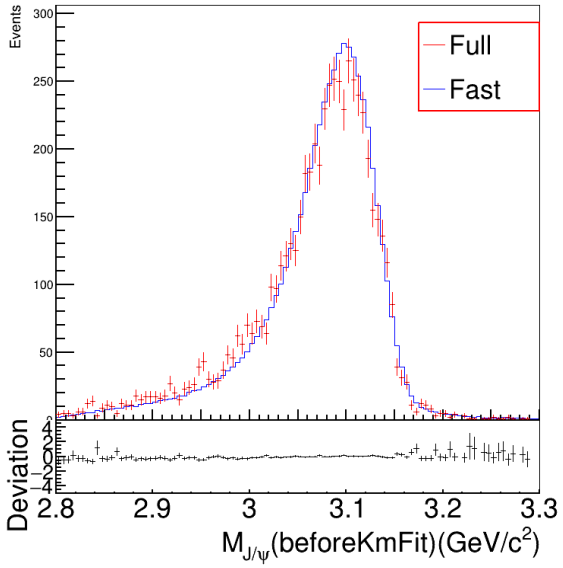
- ExtTrackCollection为空，没有径迹外推至各个子探测器的信息

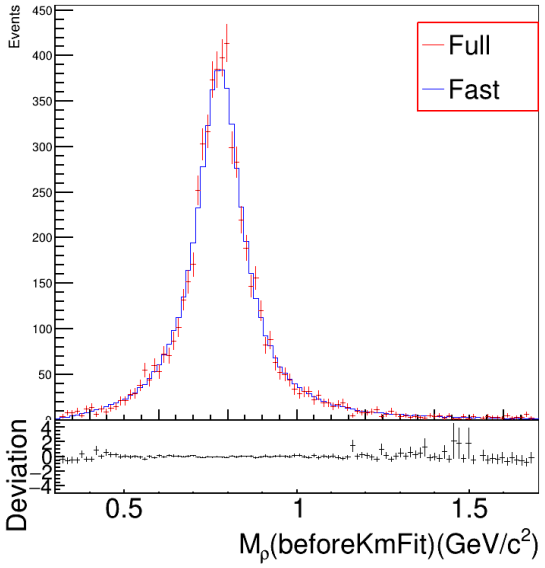
# 全重建与快模拟的对比：单粒子

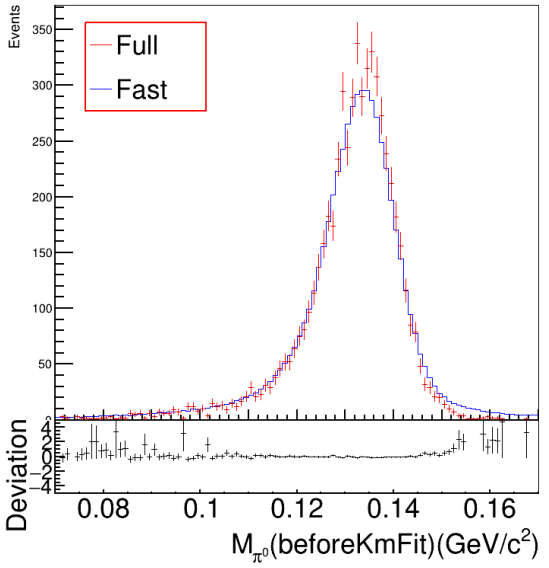


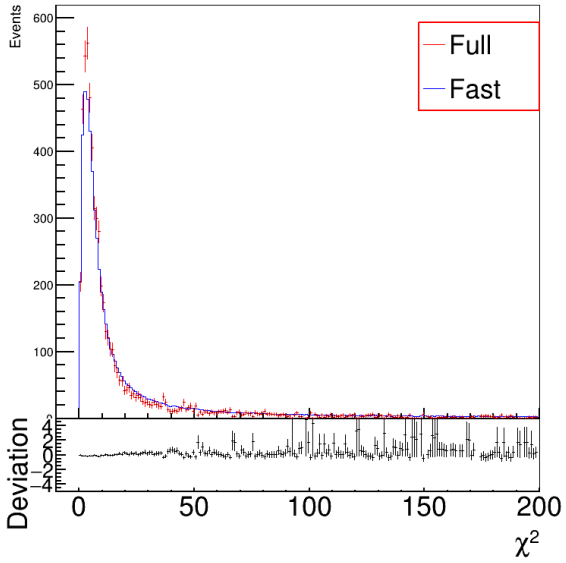



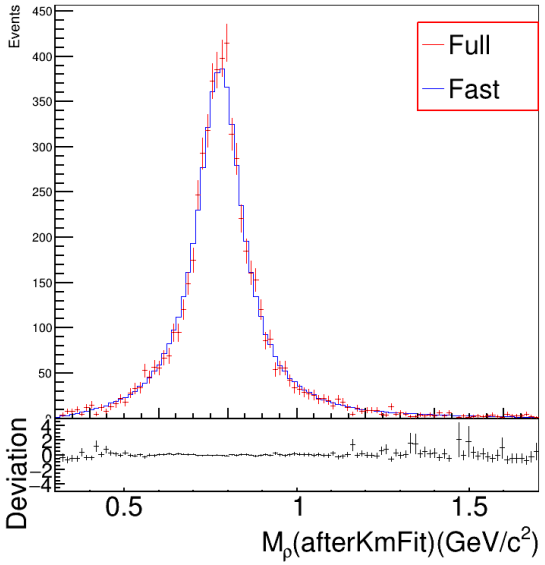
# $J/\psi \to \rho\pi^0$

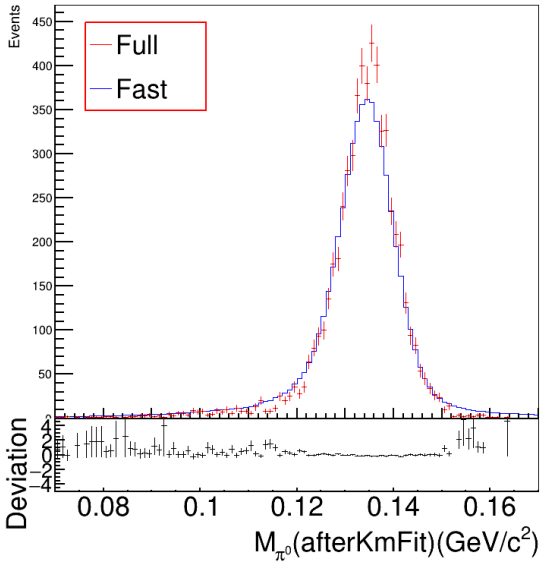


# 应用场景

- 服务于样本统计量需求非常大的物理预研究
- 对探测器响应性能的快速配置和设计
- 更多物理过程的验证

- 欢迎同学们加入到验证和开发中