Seed Data 2

19 Apr 2018, HongMin Kim

Now

- Modifying the MyGsim2Dst.cc by referring to gsim2dst2015.cc

- MPPC Test

- Radiation worker registration has been completed.

Seed Data.C

```
using namespace std;

int main(int argc,char** argv){

  int runID=atoi(argv[1]);


  TFile* t = new TFile(Form("/home/had/hmkim/work/hmkim/KLpipipi0/g2ana_wo_acc/
g2ana_%d.root",runID),"read");

  TFile* t2 = new TFile(Form("/home/had/hmkim/work/hmkim/KLpipipi0/rawMC/
MC_%d.root",runID),"read");

TTree* tr2pi0 = (TTree*)t->Get("RecTree");
TTree* tr = (TTree*)t2->Get("eventSeedTree00");
TTree* tr2 = (TTree*)t2->Get("eventTree00");

float CBARVetoEne,NCCVetoEne,CVVetoEne,CC03VetoEne,FBARVetoEne;
int MCEventID;

int CutCondition,AddCutCondition,MyVetoCondition;
double MaxShapeChisq;

tr2pi0->SetBranchAddress("MCEventID",&MCEventID);
tr2pi0->SetBranchAddress("CutCondition",&CutCondition);
tr2pi0->SetBranchAddress("AddCutCondition",&AddCutCondition);
tr2pi0->SetBranchAddress("MyVetoCondition",&MyVetoCondition);
tr2pi0->SetBranchAddress("MaxShapeChisq",&MaxShapeChisq);
```

```
  double EventStartTime,EventStartZ;
  double Pi0Pt[10];
  tr2pi0->SetBranchAddress("EventStartZ",&EventStartZ);
  tr2pi0->SetBranchAddress("EventStartTime",&EventStartTime);
  tr2pi0->SetBranchAddress("Pi0Pt",Pi0Pt);

  TFile* tnew = new TFile(Form("/home/had/hmkim/work/hmkim/KLpipipi0/seeddata/
SeedData%d.root",runID),"RECREATE");

  TTree* trnew = tr->CloneTree(0);
  TTree* trnew2 = tr2->CloneTree(0);
```

```
  int g2anaEventID;
  double RecPi0Pt;
  trnew2->Branch("g2anaEventID",&g2anaEventID,"g2anaEventID/I");
  trnew2->Branch("EventStartZ",&EventStartZ,"EventStartZ/D");
  trnew2->Branch("EventStartTime",&EventStartTime,"EventStartTime/D");
  trnew2->Branch("RecPi0Pt",&RecPi0Pt,"RecPi0Pt/D");

  int nevent=tr2pi0->GetEntries();

cout << "entry : " << nevent << endl;
```

```
  for(int ievent=0; ievent<nevent; ievent++){
    tr2pi0->GetEntry(ievent);
    cout << ievent << endl;
    g2anaEventID=ievent;
   if( (CutCondition|0xc) != 0xc) continue;
   if(  MaxShapeChisq > 4.6 ) continue;
   if( ( MyVetoCondition|0x923c ) != 0x923c) continue;

    cout << CutCondition << endl;

    tr->GetEntry(MCEventID);
    tr2->GetEntry(MCEventID);

    RecPi0Pt=Pi0Pt[0];

    trnew->Fill();
    trnew2->Fill();
  }

  trnew->Write();
  trnew2->Write();
  tnew->Close();

}
```

user_cut.cc

```
   if( (CutCondition|0xc) != 0xc) continue;
enum{ ENERGY_CUT=0,  CSI_FIDUCIAL_CUT, VERTEX_CUT,     PT_CUT,
      COLLINEAR_CUT, DISTANCE_CUT,     E_TOTAL_CUT,    E_THETA_CUT,
      E_RATIO_CUT,   PI_KINE_CUT,      SHAPE_CHI2_CUT                };
```

```
int standardCut(Pi0 const &pi0){
  Gamma const &g1 = pi0.g1();
  Gamma const &g2 = pi0.g2();

  int iCut  = 0;
  double Egamma_Min = 0.1 * 1000; //GeV to MeV
  double Egamma_Max = 2.0 * 1000; //GeV to MeV
  //  double Egamma_Max = 10.0;
  //  double R_Min =175;
  double XY_Min =150;
  double R_Max =850;
  //  double R_Max =1350;  // for step2

  double Zvert_Min =3000;
  double Zvert_Max =5000;

  double Pt_Min =0.13 * 1000; //GeV to MeV
  double Pt_Max =0.25 * 1000; //GeV to MeV

  double Acop_Min =30.; // deg
  double gDist_Min =300.; //mm
  double TotE_Min =0.5 * 1000; // GeV to MeV
  double Etheta_Min =2.5 * 1000  ; //GeV to MeV
  double Eratio_Min = 0.2;
```

```
//Energy Cut
if( g1.e()  < Egamma_Min || g2.e() < Egamma_Min ||
    g1.e() > Egamma_Max || g2.e() > Egamma_Max)
  iCut += 1<<ENERGY_CUT ;

//Fiducial in CsI
double R1 = g1.pos().perp();
double R2 = g2.pos().perp();
//  if ( R1 < R_Min || R1 > R_Max || R2 < R_Min || R2 > R_Max)
if( R1>R_Max ||  (fabs(g1.x())<XY_Min && fabs(g1.y())<XY_Min) ||
    R2>R_Max ||  (fabs(g2.x())<XY_Min && fabs(g2.y())<XY_Min) )
  iCut += 1<<CSI_FIDUCIAL_CUT;

// Vertex Cut
double RecVertZ = pi0.recZ();
if ( RecVertZ < Zvert_Min || RecVertZ > Zvert_Max)
  iCut += 1<<VERTEX_CUT ;

// PT Cut
double RecPi0Pt = pi0.p3().perp();
if ( RecPi0Pt < Pt_Min || RecPi0Pt > Pt_Max)
  iCut += 1<<PT_CUT;
```

```
// Acp_angle  Cut
double acop;
double  ct = (g1.x()*g2.x()+g1.y()*g2.y())/R1/R2;
if(TMath::Abs(ct) >= 1){
  acop = 0.0;
}
else{
  acop = TMath::Pi() - TMath::ACos(ct);
  acop = acop * 180. / TMath::Pi();
}

double Acop_angle = acop;
if (Acop_angle < Acop_Min) iCut += 1<<COLLINEAR_CUT;

//Two gamma distance
CLHEP::Hep3Vector gdist = g1.pos()-g2.pos();
double gamma_dist = gdist.mag();

if (gamma_dist < gDist_Min) iCut += 1<<DISTANCE_CUT;

//Total energy of two gammas
double E_tot = g1.e() + g2.e();
if (E_tot < TotE_Min)  iCut += 1<<E_TOTAL_CUT;

// In_ang cut : Odd-pair cut
double  In_Ang1 = TMath::ATan(R1/(g1.z()-pi0.recZ()));
In_Ang1 = In_Ang1 * 180 / TMath::Pi();
double  Etheta1 = g1.e() * In_Ang1;
double  In_Ang2 = TMath::ATan(R2/(g2.z()-pi0.recZ()));
In_Ang2 = In_Ang2 * 180 / TMath::Pi();
double  Etheta2 = g2.e() * In_Ang2;


if (Etheta1 < Etheta_Min || Etheta2 < Etheta_Min)
  iCut += 1<<E_THETA_CUT;
```

```
  // Pi0 Kinematic Cut
  if(pi0kine_cut(pi0))  {
    iCut += 1<<PI_KINE_CUT;
  }
  return iCut;
}

bool cutline(double x1, double y1, double x2, double y2,
                      double var1, double var2)
{
  double slope = ( y1 - y2 ) / ( x1 - x2 );
  double intercept = y1 - slope * x1 ;
  double y = slope * var1 + intercept;
  if ( var2 > y ) return kTRUE; // above the line
  return kFALSE;
}
```

```
bool pi0kine_cut(Pi0 const &pi0)
{
  double cpi0pt=pi0.p3().perp()*1e-3;//RecPi0Pt*1e-3;
  double cpi0pz=pi0.p3().z()*1e-3;//RecPi0Pz*1e-3;
  double cpi0recz=pi0.recZ()*1e-1;//RecVertZ*1e-1;
  double cpi0e=pi0.e()*1e-3;//RecPi0E*1e-3;

  bool flag = kFALSE;
  double pratio = cpi0pt/cpi0pz;
  if (( cpi0recz < 400 )&&( pratio < 0.1 )) return kTRUE;
  if (( cpi0recz > 400 )&&
      !( cutline(400., 0.1, 500., 0.15, cpi0recz, pratio) )) return kTRUE;
  if ( !(cutline(300., 0.8, 500., 0.4, cpi0recz, cpi0e)) ) return kTRUE;
  if ( (cutline(300., 0.2, 500., 0.34, cpi0recz, pratio)) ) return kTRUE;
  return kFALSE;

}
```

```
double getCsiThreshold(double distance){
  double constant = 3.73275;
  double slope = 0.0074758;
  double threshold = exp(constant-slope*distance);
  if(threshold>10) return 10;
  if(threshold<1.5) return 1.5;
  return threshold;
}
```

```
int csiVetoCut(E14GNAnaDataContainer const &data,Pi0 const &pi){
  bool isVeto = false;

  for(int icsi = 0; icsi < data.CsiNumber; icsi++){
    int id = data.CsiModID[icsi];
    double edep  = data.CsiEne[icsi];

    bool find = false;
    double cx[2]={0},cy[2]={0};
    for(int igam=0; igam<2 ; igam++){
      Gamma const &gam = (igam==0)? pi.g1() : pi.g2();
      cx[igam] = gam.coex();
      cy[igam] = gam.coey();
      for(int i=0; i<gam.clusterIdVec().size()&&!find; i++){
        if(gam.clusterIdVec().at(i)==id)  find = true;
      }
    }

    if(find) continue;
    double x,y,w;
    CsiMap::getCsiMap()->getXYW(id,x,y,w);
    double dist = std::min(sqrt(pow(x-cx[0],2)+pow(y-cy[0],2)),
                           sqrt(pow(x-cx[1],2)+pow(y-cy[1],2)));

    if(edep > getCsiThreshold(dist)){
      isVeto = true;
      //      break;
    }
  }
  if(isVeto) return 1<<DigiReader::CSI;
  return 0;
}
```

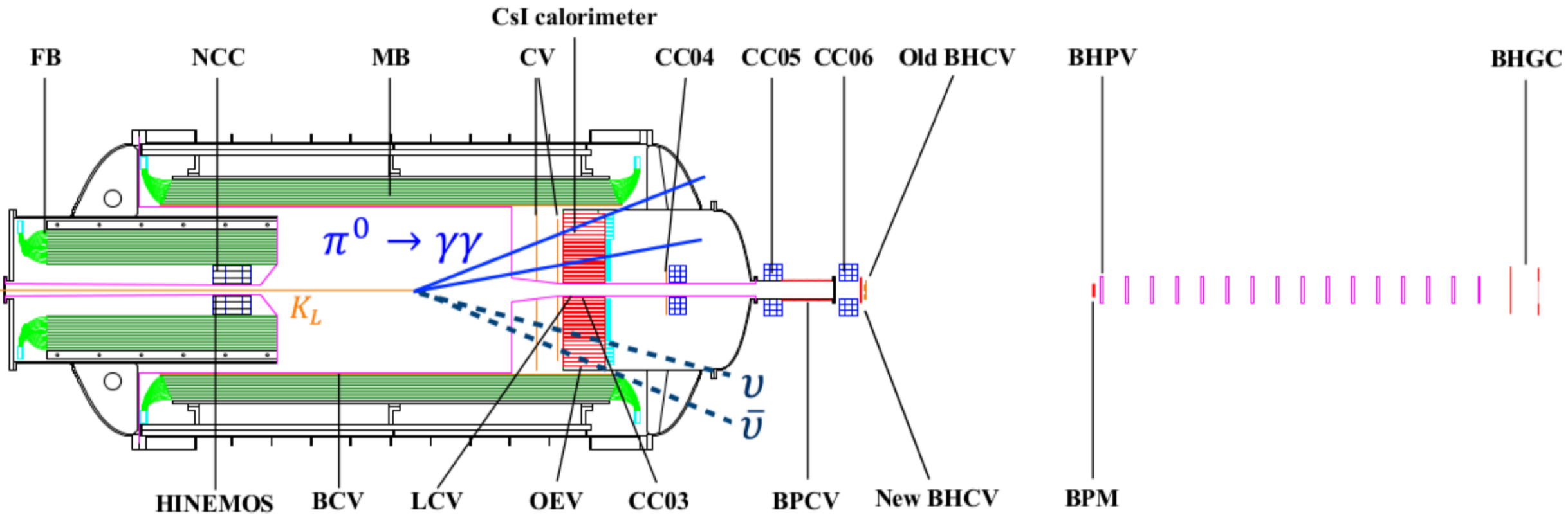

Figure 2.1: Schematic view of KOTO detector in 2015.

MTBasicParameters.h

```
   if( ( MyVetoCondition|0x923c ) != 0x923c) continue;
typedef enum { CC01=0, CC02,    CC03,    CC04,  CC05,
               CC06,   CBAR,    FBAR,    CSI,   BHPV,
               CV,     BCV,     BHCV,    NCC,   OEV,
               LCV,    BPCV,    newBHCV, BHGC,  IB,
               IBCV,   MBCV,    COSMIC,  LASER, ETC,  TRIGGERTAG } DetectorID;
```

```
  const int DetNChannels[nDetectors] =
    { 0,    0,    32,  64,   60,
      60,   128,  32,  2716, 34,
      184,  64,   8,   204,  44,
      4,    4,    48,  8,    64,
      64,   16,   16,  4,    0,   6 };
  const int DetNModules[nDetectors] =
    { 0,    0,    16,  62,   58,
      58,   64,   32,  2716, 34,
      92,   32,   8,   204,  44,
      4,    4,    48,  4,    32,
      32,   16,   16,  4,    0,   6 };
  const Bool_t is125MHzDetector[nDetectors] =
    { true,  true,  true,  true,  true,
      true,  true,  true,  true,  false,
      true,  true,  false, true,  true,
      true,  true,  false, false, false,
      true,  true,  true,  true,  true,  true };
  const Bool_t is500MHzDetector[nDetectors] =
    { false, false, false, false, false,
      false, false, false, false, true,
      false, false, true,  false, false,
      false, false, true,  true,  true,
      false, false, false, false, false, false };
```

```
const Float_t AccTimeShift[nDetectors] =
   { 0,       0,       40.487,   15.40,    -13.1,
     -10.6,   51.34,   49.09,    30.6098,  -67.3,
     65.46,   50.89,   -82,      34.059,   27.02,
     24.91,   25.44,   -81.2,    -64.7,    -20.0,
     29.29,   29.29,   0,        0,        0,        0}; // change by shiomi 2017/11/30

 const Float_t VetoEneThre[nDetectors] =
   { 0,       0,     3.,          3.,   3.,
     3.,      2.,    2.,          0.,   2.5,
     0.2,     1.,    0.3,         2.,   2.,
     0.6,     1.,    884.e-6/4,   2.5,  2.,
     1.,      1.,    0,           0,    0 ,   0};

 const Float_t VetoTiming[nDetectors] =
   { 0,       0,      32.2,    4.31,    -24.3,
     -21.8,   38.8,   35.08,   0,       -76.2,
     53.7,    37.9,   -94.5,   9.32,    19.5,
     14.9,    21.4,   0,       -77.4,   -40.0,
     15.0,    10.0,   0,       0,       0,        0 }; // change by shiomi 2017/11/30

 const Float_t VetoWidth[nDetectors] =
   { -1,        -1,          15.*2,    15.*2,    15.*2,
     15.*2,     30.*2,       20.*2,    -1.,      7.5*2,
     40.*2,     30.*2,       7.5*2,    20.*2,    10.*2,
     15.*2,     12.*2,       -1,       7.5*2,    30.*2,
     30.*2,     30.*2,       -1,       -1,       -1 ,    -1 };
```