OAK RIDGE National Laboratory

EIC

Detector Configurations in Simulations

ECCE Simulation Workshop

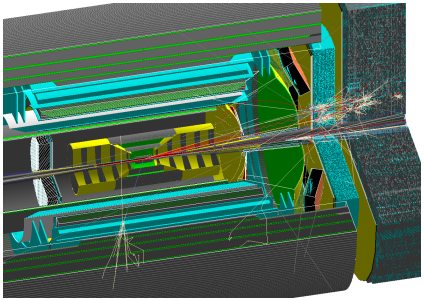
April 2, 2021

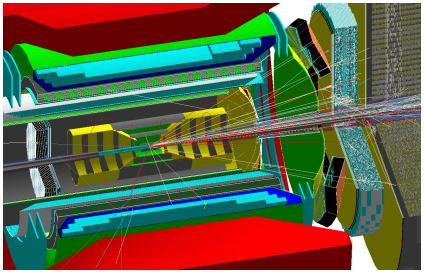
Friederike Bock & Nicolas Schmidt
for the ORNL Relativistic Nuclear Physics Group

F. Bock, T. Cormier, L. Cunqueiro, M. Demarteau, R. Ehlers, M. Fasel, E. Glimos, H. Hassan, F. Jonas, C. Loizides, J. Osborn, M. Poghosyan, K. Read, A. Russu, J. Schambach, N. Schmidt, S. Sorenson

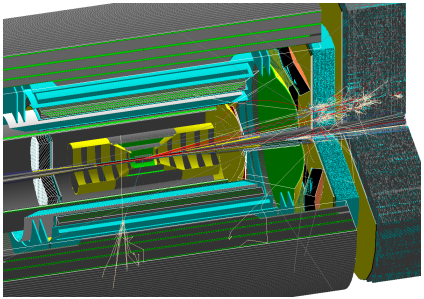
What we will cover today - Overview

- How to create a full detector setup with existing sub-detectors?
- How to simulate the full detector?
- How to adapt the geometry of a sub-detector system?
- How to create a new sub-detector system with existing objects?
- How to modify the tracking algorithm?
- How to implement a new sub-detector system?
- How to evaluate the output?





How to create a full detector setup with existing sub-detectors?



Several examples of full detector setups exist:

- **fun4all_macros/detectors/EICDetector**
- **fun4all_eicmacros/detectors/JLeic**
- **fun4all_eicmacros/detectors/Modular**
- **fun4all_eicmacros/detectors/ModularBeast**

Will use **fun4all_eicmacros/detectors/Modular** for further explanations

Common structure:

- **Fun4All_G4_XXX.C**
  - Steering of detector & event generator setup
  - 4 main parts:

a) generator setup
b) enable/disable sub-detectors & settings
c) reconstruction setup
d) setup event evaluators

- **G4Setup_XXX.C**
  - Geometry intialization of the sub-dectors enabled in **Fun4All_G4_XXX.C**
- **DisplayOn.C**
  - Steering of G4-visualization

Detector macro - External Parameters

Example Fun4All_G4_FullDetectorModular.C:

```
int Fun4All_G4_FullDetectorModular(
  const int nEvents                 = 1,
  const double particlemomMin       = -1,
  const double particlemomMax       = -1,
  TString specialSetting            = "ALLSILICON-FTTLS3LC-ETTL-CTTL",
  TString generatorSettings         = "el0p250MB",
  const string &inputFile           = "https://www.phenix.bnl.gov/WWW/publish/....",
  const string &outputFile          = "G4EICDetector.root",
  const string &embed_input_file    = "https://www.phenix.bnl.gov/WWW/publish/....",
  const int skip                    = 0,
  const string &outdir              = ".")
```

- `nEvents`: number of events to be generated, for interactive running put -1
- `particlemomMin`, `particlemomMax`: steering of single particle gun setup
  if `particlemomMin = particlemomMax` monoenergetic particle else momentum range for single particle generator
- `generatorSettings`: steering of generator, i.e.
  `"SinglePion"`, `"el0p250MB"`, `"pTHard5"`, `"FPartTrigg"`, `"FJetTrigg"`
  can be composed string

- `outputFile`, `outdir`: beginning of output file name and storage location
- `inputFile`, `embed_input_file`: externally generated event files, mainly used for HEP-MC usage or fully generated DST tree
- `skip`: skipping event generation and G4 simulation, reading DST tree

- `specialSetting`: steering of detector setup, i.e.
  `"ALLSILICON-FTTLS3LC-ETTL-CTTL"` - LBL Si-tracker + TTL + calorimeters
  `"MVTX-EGEM-FGEM"` - MVTX + TPC + GEMs + calorimeters
  `"BARRELV4-FSTV42-FTTLS3LC-ETTL-CTTL"` - LANL Si-tracker V4 + TTL + calorimeters
  can also steer versions of same detector i.e. `"BARRELV4"`, `"FTTLSE1LC"`, `"HC2x"` partially parsed by `void ParseTString(TString &specialSetting)`

same usage for Fun4All_G4_FullDetectorModularBeast.C

Detector macro - Generator setups

```
if (Input::SIMPLE){   // single particle energy range
  if (generatorSettings.Contains("SimplePion"))
    INPUTGENERATOR::SimpleEventGenerator[0]->add_particles("pi-", 1);
  ...
  INPUTGENERATOR::SimpleEventGenerator[0]->set_pt_range(particlemomMin, particlemomMax);
}
if(particlemomMin>-1 && particlemomMax == -1){ // fixed energy
  PHG4ParticleGenerator *gen = new PHG4ParticleGenerator("PGENERATOR");
  gen->set_name("pi-");
  if(particlemomMin > -1)
    gen->set_mom_range(particlemomMin, particlemomMin);
  ...
  se->registerSubsystem(gen);
}
if (Input::PYTHIA6){ // pythia6
  if (generatorSettings.CompareTo("e10p250MB") == 0)
    INPUTGENERATOR::Pythia6->set_config_file(string(getenv("CALIBRATIONROOT"))
                                             + "/Generators/phpythia6_ep.cfg");
  ...
  if (generatorSettings.Contains("FPartTrigg")){
    PHPy6ParticleTrigger *ptrig = new PHPy6ParticleTrigger();
    ptrig->SetPtLow(1);
    ptrig->SetEtaHighLow(1,5);
    INPUTGENERATOR::Pythia6->register_trigger(ptrig);
  }
  if (generatorSettings.Contains("FJetTrigg")){
    PHPy6JetTrigger *trig = new PHPy6JetTrigger();
    trig->SetEtaHighLow(1,5);
    if (generatorSettings.Contains("pTHard5"))
      trig->SetMinJetPt(5);
    ...
    trig->SetJetR(0.7);
    INPUTGENERATOR::Pythia6->register_trigger(trig);
  }
}
```

Multiple options for Event generations

- `INPUTGENERATOR::SimpleEventGenerator` simple event generator with monocromatic particles
- `INPUTGENERATOR::VectorMesonGenerator` vector meson generator
- `INPUTGENERATOR::Gun`, `PHG4ParticleGenerator` single particle gun
- `INPUTGENERATOR::Pythia6` full event generator using Pythia6, different e-p setups
- `INPUTGENERATOR::Pythia8` full event generator using Pythia8, different e-p setups
- `INPUTGENERATOR::SARTRE` event generator for exclusive diffractive vector meson production and DVCS in ep and eA collisions based on the dipole model
- `Input::HEPMC` reading of pregenerated events, enables pile-up studies, crossing angle, ...

Possibility to trigger on jets or single particles in full event generators:

- `PHPy6ParticleTrigger`, `PHPy8ParticleTrigger` trigger on specific particle type, η region or minimum momentum
- `PHPy6JetTrigger`, `PHPy8JetTrigger` trigger on particle level jet of siz R in η region with minimum jet momentum

Detector macro - Detector enabling

```
// barrel tracker (LANL)
if (specialSetting.Contains("BARREL"))
  Enable::BARREL = true;
if(specialSetting.Contains("FST"))
  Enable::FST = true;

// all silicon tracker version (LBL)
if(specialSetting.Contains("ALLSILICON")){
  Enable::ALLSILICON = true;
  Enable::ALLSILICON_ABSORBER = true;
}
// LGAD layers
if(specialSetting.Contains("FTTL"))
  Enable::FTTL = true;
if(specialSetting.Contains("ETTL"))
  Enable::ETTL = true;
if(specialSetting.Contains("CTTL"))
  Enable::CTTL = true;

// mvtx/tpc tracker
if(specialSetting.Contains("MVTX")){
  Enable::MVTX = true;
  Enable::TPC = true;
}
...
if(specialSetting.Contains("FEMCSTANDALONE") )
  Enable::FHCAL = false;
Enable::FHCAL_VERBOSITY = 1;
//  Enable::FHCAL_ABSORBER = true;
Enable::FHCAL_CELL = Enable::FHCAL && true;
Enable::FHCAL_TOWER = Enable::FHCAL_CELL && true;
Enable::FHCAL_CLUSTER = Enable::FHCAL_TOWER && true;
Enable::FHCAL_EVAL = Enable::FHCAL_CLUSTER && false;
```

Several different sub-detector systems implemented as G4_XXX.C can be found in fun4all_eicmacros/common and fun4all_macros/common

- Silicon detectors: MVTX, ALL-SILICON, TTL, Barrel_EIC, FST, VTX_JLeic, CTD, INTT
- other Tracking: FGEM_fsPHENIX, GEM_EIC, Micromegas, TPC_EIC, Gem_JLeic
- PID detectors: DIRC, DIRC_JLeic, DRich_JLeic, Aerogel, TTL, PSTOF, RICH
- Calorimeters: EEMC, EndCap_Electron_JLeic, CEMC(_Spacal, _Albedo, _EIC), HcalIn, HcalOut, Barrel_Hcal_JLeic, FEMC, FHCAL, EndCap_Hadron_JLeic
- Magnets: Magnet (Babar), Magnet_Cleo, Magnet_Beast
- forward detectors: Bbc, EPD
- Beamline: Pipe_EIC, BeamLine_JLeic, hFarFwdBeamLine_ip6_EIC

implemented in the different detector setups
detectors can partially overlap, setup exclusion need to be defined
full G4 setup created in G4Setup_XXX.C
i.e G4Setup_ModularDetector.C

Create a detector from existing objects

Example: G4_TTL_EIC.C adapted from Barrel_EIC & FST macros

```
void CTTLSetup(PHG4Reco *g4Reco, TString cttloption = ""){
  const double cm = PHG4Sector::Sector_Geometry::Unit_cm();
  const double mm = .1 * cm;
  const double um = 1e-3 * mm;
  make_barrel_layer("CTTL_0", g4Reco, 92,  180, 85*um);
  make_barrel_layer("CTTL_1", g4Reco, 114.7,  180, 85*um);
}

int make_barrel_layer(string name, PHG4Reco *g4Reco,
                      double radius, double halflength, double tSilicon){

  const int nSubLayer = 7;
  const double cm = PHG4Sector::Sector_Geometry::Unit_cm();
  const double mm = .1 * cm;
  const double um = 1e-3 * mm;
  string layerName[nSubLayer] = {"SiliconSensor", "Metalconnection", "HDI", "Cooling",
                                 "Support1", "Support_Gap", "Support2"};
  string material[nSubLayer] = {"G4_Si", "G4_Al", "G4_KAPTON", "G4_WATER",
                                "G4_GRAPHITE", "G4_AIR", "G4_GRAPHITE"};
  double thickness[nSubLayer] = {tSilicon , 15 * um, 20 * um, 100 * um,
                                 50 * um, 1, 50 * um};

  double max_bh_radius = 0.;
  PHG4CylinderSubsystem* cyl;
  double currRadius = radius;
  for (int l = 0; l < nSubLayer; l++) {
    cyl = new PHG4CylinderSubsystem(name + "_" + layerName[l],l);
    cyl->SuperDetector(name);
    cyl->set_double_param("radius", currRadius);
    cyl->set_double_param("length", 2.0 * halflength);
    cyl->set_string_param("material", material[l]);
    cyl->set_double_param("thickness", thickness[l]);
    if (l == 0) cyl->SetActive();  //only the Silicon Sensor is active
    cyl->OverlapCheck(true);
    g4Reco->registerSubsystem(cyl);
    currRadius = currRadius+thickness[l];
  }
```

← **Setup the full Barrel detector**

← **function create your barrel layer**

← **define how many sublayers you have**

← **define materials of sub-layers and depth**

← **creating 1 layer of the CTTL out of the different material of sublayers**

← **cylinder geometry created from** `PHG4CylinderSubsystem`
one of the main detector building components
analogously for disk dectors i.e. in forward direction `PHG4SectorSubsystem` **see same macro**
`int make_forward_station(...)`

← **don't forget the overlap check**

Including the detector in the tracking

Example: Including CTTL in G4_Tracking_Modular.C macro)

```
void Tracking_Reco(TString specialSetting = ""){

  PHG4TrackFastSim *kalman = new PHG4TrackFastSim("PHG4TrackFastSim");
  kalman->set_sub_top_node_name("TRACKS");
  kalman->set_trackmap_out_name(TRACKING::TrackNodeName);

  float posResImp = sqrt(12);
  if (specialSetting.Contains("ACLGAD"))
    posResImp = sqrt(256);
  // central barrel
  if (Enable::CTTL){
    float pitch=500e-4;
    float res   = pitch/posResImp;
    int nlayer  = 2;
    for (int i = 0; i < nlayer; i++){
      kalman->add_phg4hits(Form("G4HIT_CTTL_%d", i),
                           PHG4TrackFastSim::Cylinder,
                           999,  //  const float radres  *ignored in barrel detector*
                           res,  //  const float phires,
                           res,  //  const float lonres, *ignored in plane detector*
                           0.95, //  const float ef \begin{picture}(340,220)
                           0);   //  const float noise
    }
    kalman -> add_cylinder_state("CTTL_0", 92);
    kalman -> add_cylinder_state("CTTL_1", 114.7);
  }
}
void Tracking_Eval(const std::string &outputfile, TString specialSetting = ""){
  // save tracking planes in forward direction for single layer studies
  if (Enable::CTTL){
    int nlayer  = 2;
    for (int l = 0; l < nlayer; l++)
      fast_sim_eval->AddProjection(Form("CTTL_%d",l));
  }
}
```

← **Kalman filter for tracks (there can be multiple, so check the names)**

← **What is the spatial resolution of your detectors**

← **Adding hits for the different CTTL layers**

← **Define geometry of FastTracking layer**
`PHG4TrackFastSim::Cylinder` or `PHG4TrackFastSim::Vertical_Plane`

← **Set resolutions in R, φ, Z, efficiencies, noise level**, ignores Z for plane detectors & R for cylinders

← **Adding cylinder or plane state for each layer for exact projection output on layer**

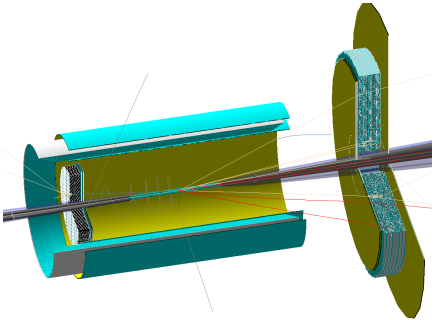
← **Don't forget to add them to the** `Tracking_Eval()` **if you want the projections for PID estimates or other studies**

Looking at the detector and simulating single events

Some useful G4 visualization commands:
can be executed in root-session with g4->ApplyCommand("...")

- `/vis/viewer/zoom 2` zoom level
- `/vis/viewer/set/viewpointThetaPhi 260 -40` defines from where you are looking at the detector
- `/vis/viewer/panTo 50 0 cm` moves center of visulization in x and y on display
- `/vis/viewer/addCutawayPlane 0 0 0 m 1 0 0` cuts out planes (you can have up to 3)
- `/vis/viewer/clearCutawayPlanes` removes cutaway planes
- `/vis/viewer/set/background white` sets bg-color
- `/vis/modeling/trajectories/create/drawByParticleID (drawByCharge)`
  `/vis/modeling/trajectories/drawByParticleID-0/set e+ steelblue`
  you can set color filters by Charge or particle ID
  (e+/-,pi+/-,kaon+/-,proton+/-,neutron,gamma,neutrino..) many many colors available
- `/vis/filtering/trajectories/create/particleFilter`
  `/vis/filtering/trajectories/particleFilter-0/add neutron`
  `/vis/filtering/trajectories/particleFilter-0/invert true`
  will display all particles in the event except neutrons
- `/vis/filtering/trajectories/create/attributeFilter`
  `/vis/filtering/trajectories/attributeFilter-0/setAttribute IMag`
  `/vis/filtering/trajectories/attributeFilter-0/addInterval 10 MeV 1000 GeV`
  will display all particle with an energy between 10MeV and 1000 GeV

adding filters might be cumbersome, and doesn't necessarily always work (max is 3). Don't get frustrated!
There are much more options have a look at the GEANT4 tutorials.



Running interactively in root

```
.x Fun4All_G4_FullDetectorModular.C(-1,-1,-1,"FTTL3LC-ETTL-CTTL",
                                    "e10p250MB")
PHG4Reco *g4 = DisplayOn();
Fun4AllServer *se = Fun4AllServer::instance();
se->run(1);
g4->ApplyCommand("/vis/viewer/refresh");
```

How to implement a detector system - Overview

Overview of necessary classes/headers in coresoftware/simulation/g4simulation/g4detectors:

- **PHG4MyDetSubsystem.cc/h**
  - main function: `InitRunSubsystem()`
    → defines output nodes (e.g. for Geant4 hits)
    → calls the constructors for the three other classes
  - initialize global variables with default values here
- **PHG4MyDetDetector.cc/h**
  - contains the Geant4 detector geometry
  - build detector in world volume within `ConstructMe()`
    → place `G4VSolids` in `G4LogicalVolumes` via `G4PVPlacement/G4PVReplica/...`
    → when placing $>$ 1000 objects, replicas are recommended
- **PHG4MyDetDisplayAction.cc/h**
  - define colors and visibility for Geant4 viewer for individual components
- **PHG4MyDetSteppingAction.cc/h**
  - processes Geant4 hits in the detector materials
    → fills `PHG4HitContainer` with all infos needed for further processing (e.g. by a clusterizer)
  - main function: `UserSteppingAction()`
- **Makefile.am**
  → add your headers and classes here to be included in the compilation

```
#how to compile the newly added or changed code in e.g. g4detectors
#everything below takes place within singularity!
cd coresoftware/simulation/g4simulation/g4detectors
source /cvmfs/eic.opensciencegrid.org/gcc-8.3/opt/fun4all/core/bin/eic_setup.sh -n
#define the install location for the built code
export MYINSTALL=~/install
source /cvmfs/eic.opensciencegrid.org/gcc-8.3/opt/fun4all/core/bin/setup_local.sh $MYINSTALL
autogen.sh --prefix=$MYINSTALL
make install
```

It is easiest to start with a copy of an existing detector!

How to implement a detector system (1)

General remarks about PHG4MyDetSubsystem.cc/h

```
int PHG4MyDetSubsystem::Init(PHCompositeNode* topNode)
{
  PHNodeIterator iter(topNode);
  PHCompositeNode* dstNode =
    dynamic_cast<PHCompositeNode*>(iter.findFirst("PHCompositeNode", "DST"));
  // create display settings before detector
  m_DisplayAction = new PHG4MyDetDisplayAction(Name());
  // create detector
  m_Detector = new PHG4MyDetDetector(this, topNode, Name());
  /* set global settings here for PHG4MyDetDetector (e.g. Verbosity) */
  if (active)
  {
    set<string> nodes;
    // create hit output node
    ostringstream nodename;
    nodename << "G4HIT_" << detector_type;
    PHG4HitContainer* scintillator_hits =
        findNode::getClass<PHG4HitContainer>(topNode, nodename.str());
    if (!scintillator_hits)
    {
      scintillator_hits = new PHG4HitContainer(nodename.str());
      PHIODataNode<PHObject>* hitNode =
        new PHIODataNode<PHObject>(scintillator_hits, nodename.str(), "PHObject");
      dstNode->addNode(hitNode);
      nodes.insert(nodename.str());
    }
    /* ... same as above also for absorber */

    // create stepping action
    m_SteppingAction =
        new PHG4MyDetSteppingAction(m_Detector,absorber_active);
  }
  return 0;
}
```

← **initialize DisplayAction (for Geant4 visualization)**

← **initialize your detector (the geometry)**
this includes setting global properties, like verbosity, configuration files, and possibly more

← **define hit nodes and containers**

← **initialize stepping action (for detector hit association)**

How to implement a detector system (2)

General remarks about building detector geometry in PHG4MyDetDetector.cc/h

```
// define base mother volume of detector (a large box, cylinder, ...)
// and its logical volume
G4VSolid* mydet_envelope_solid = new G4Cons/G4Box/...(/* dimensions */);
G4LogicalVolume* mydet_envelope_log =
    new G4LogicalVolume(mydet_envelope_solid, Air,
                        G4String("mydet_envelope"), 0, 0, 0);
// add logical volumes to display action for Geant4 viewer
 m_DisplayAction->AddVolume(mydet_envelope_log, "mydetEnvelope");
// logical volumes need to be placed in the "world"
 new G4PVPlacement(0 /*rotation*/, G4ThreeVector(x,y,z /*position*/)),
                   mydet_envelope_log, name_envelope.str().c_str(),
                   logicWorld, 0, false, OverlapCheck());

/*.....*/

// further detector elements are placed in overlying logical mother volume
G4VSolid* solid_scintillator = new G4Box(/* dimensions */);
G4Material* material_scintillator =
    G4Material::GetMaterial(m_Params->get_string_param("scintillator"));

G4LogicalVolume* logic_scint = new G4LogicalVolume( solid_scintillator,
                                                    material_scintillator,
                                                    "mydet_scintillator_plate_logic",
                                                    0, 0, 0);
m_DisplayAction->AddVolume(logic_scint, "Scintillator");
new G4PVPlacement(0, G4ThreeVector(x,y,z /*position relative to mother center*/),
                  logic_scint,
                  name_scintillator,
                  overlying_mother_volume_logic,
                  0, 0, OverlapCheck());
```

← first create a mother volume that encloses full detector subsystem (a box, cone, cylinder, doughnut,...)

- **The default way for any detector part is as follows:**
  - **create a** `G4VSolid` **(**`G4Box`**,** `G4Cons`**, ...)**
  - **create a** `G4LogicalVolume` **from this solid (with material - here just air)**
  - **add logical volume to display action if you want it available in the Geant viewer**
  - **place logical volume in an overlying logical mother volume (here** `logicWorld`**)**

← all other detector elements follow the same approach

- **The default way for any detector part is as follows:**
  - **create a** `G4VSolid`**: here** `solid_scintillator`
  - **create a** `G4LogicalVolume`**: here** `logic_scint`
  - **add logical volume to display action**
  - `G4PVPlacement` **of** `logic_scint` **at position x,y,z within mother volume** `overlying_mother_volume_logic`

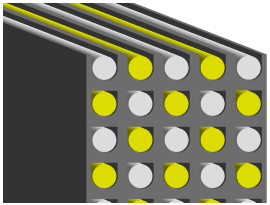
How to implement a detector system (3)

General remarks about PHG4MyDetDisplayAction.cc/h

```
void PHG4MyDetDisplayAction::ApplyDisplayAction(G4VPhysicalVolume *physvol)
{
  for (auto it : m_LogicalVolumeMap)
  {
    G4LogicalVolume *logvol = it.first;
    if (logvol->GetVisAttributes())
    {
      continue;
    }
    G4VisAttributes *visatt = new G4VisAttributes();
    visatt->SetVisibility(true);
    visatt->SetForceSolid(true);
    m_VisAttVec.push_back(visatt);  // for later deletion
    if (it.second == "Absorber"){
      visatt->SetColour(G4Colour::Gray());
    } else if (it.second == "mydetEnvelope"){
      visatt->SetVisibility(false);
    } else if (it.second == "Scintillator"){
      visatt->SetColour(G4Colour::White());
    } else if (it.second == "Cherenkov"){
      visatt->SetColour(G4Colour::Yellow());
    } else if (it.second == "SingleTowerAbsorber"){
      visatt->SetColour(G4Colour::Gray());
    } else if (it.second == "SingleTowerScintillator"){
      visatt->SetColour(G4Colour::Cyan());
      visatt->SetVisibility(false);
    } else {
      cout << "unknown_logical_volume_" << it.second << endl;
      gSystem->Exit(1);
    }
    logvol->SetVisAttributes(visatt);
```

This class allows you to set custom colors for all logical volumes

- **the name here must coincide with the string passed to** `m_DisplayAction→AddVolume(...)` **in** `PHG4MyDetDetector.cc`
- **default colors are defined [here]**
- **custom colors can be made**
- **visibility of logical volumes can be set via** `"visatt→SetVisibility(false);"`
- **the code will crash if you added a volume to m_DisplayAction but its color is not defined here!**



Example display:

← white: "Scintillator" fiber

← yellow: "Cherenkov" fiber

← grey: iron "Absorber" material

How to implement a detector system (4)

General remarks about PHG4MyDetSteppingAction.cc/h

```
bool PHG4ForwardDualReadoutSteppingAction::UserSteppingAction(const G4Step* aStep, bool)
{
  G4TouchableHandle touch = aStep->GetPreStepPoint()->GetTouchableHandle();
  G4VPhysicalVolume* volume = touch->GetVolume();
  if(!(detector_->IsInMyDet(volume)) return false;
  /* Get energy deposited by this step */
  G4double edep = aStep->GetTotalEnergyDeposit() / GeV;
  G4double eion = (aStep->GetTotalEnergyDeposit() -
                   aStep->GetNonIonizingEnergyDeposit()) / GeV;
  /* Get pointer to associated Geant4 track */
  const G4Track* aTrack = aStep->GetTrack();
  switch (prePoint->GetStepStatus())
    {
    case fGeomBoundary:
    case fUndefined:
      if (!hit)
      {
        hit = new PHG4Hitv1();
      }
      hit->set_scint_id(tower_id);
      FindTowerIndexFromPosition(prePoint, idx_j, idx_k);
      /* Set hit location (tower index) */
      hit->set_index_j(idx_j);
      hit->set_index_k(idx_k);
 /* ..... */
 /* Update exit values- will be overwritten with every step until
       * we leave the volume or the particle ceases to exist */
    hit->set_x(1, postPoint->GetPosition().x() / cm); //similar for y,z
    hit->set_t(1, postPoint->GetGlobalTime() / nanosecond);
    /* sum up the energy to get total deposited */
    hit->set_edep(hit->get_edep() + edep);
    hit->set_eion(hit->get_eion() + eion);
    if (whichactive > 0)
      hit->set_light_yield(hit->get_light_yield() + light_yield);
```

← **this class processes geant steps (particle propagation points at which physics processes are invoked)**

- **the step information is used to fill** `PHG4Hitv1()` **objects**
  → energy, position, time, track information
- **for further processing, information like tower indices can be set (as required by the clusterizer)**
- **it is easiest to use the SteppingAction of another similar detector as starting point**

Using the new detector system

Necessary changes e.g. in fun4all_eicmacros/detectors/Modular/Fun4All_G4_FullDetectorModular.C:

- **new macro needed in fun4all_eicmacros/common called e.g. G4_MyDet.C**
  - contains steering booleans in "`namespace Enable`"
  - contains its own namespace, e.g. "`namespace MYDET`"
  - contains `MyDetInit()` function to check/set global properties
  - contains `MyDetSetup()` function where `PHG4MyDetSubsystem` is initialized and registered as subsystem
    → can also contain multiple intstances of `PHG4MyDetSubsystem` (see `G4_TTL_EIC.C` for example)
  - can contain further components, e.g.:
    → tower builder component (for calorimeter)
    → clusterizer component

Make sure to check for overlaps with existing detector systems!

Deactivate them, if needed!

- **changes to G4Setup_ModularDetector.C**
  - include `G4_MyDet.C`
  - add "`if (Enable::MYDET) MyDetInit();`" to `G4Init()`
  - add "`if (Enable::MYDET) MyDetSetup(g4Reco);`" to `G4Setup()`
  - add G4HIT nodes/containers to `ShowerCompress()` and `DstCompress()`
  - can also contain `MyDet_Eval` function
    → contains e.g. `CaloEvaluator` or `PHG4TrackFastSimEval`
- **changes to Fun4All_G4_FullDetectorModular.C**
  - set Enable namespace booleans for MYDET to true (e.g. `Enable::MYDET = true;`)
  - call possible components (tower finder, clusterizer, ...)
    → e.g. `if (Enable::MYDET_TOWER) MyDet_Towers();`

Evaluating your Output (1)

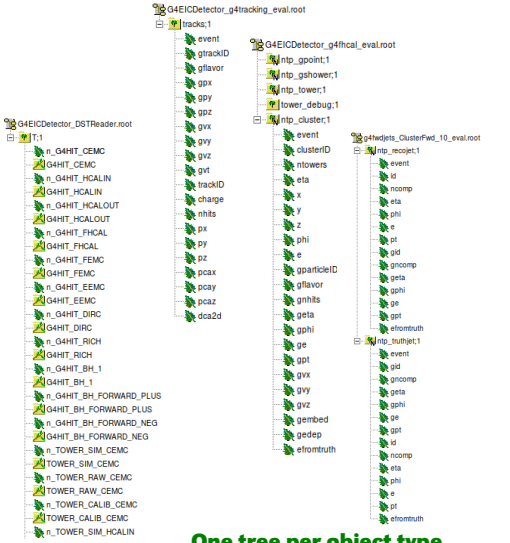
"Raw" and full output:

- PHG4DSTReader
  - provides the full NODE tree output (hits, towers, etc) with the respective Fun4All object types
  - VERY large output and not well suited for further processing

Object-based eval tasks:

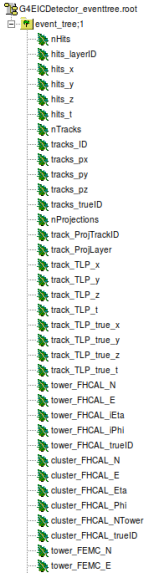
- PHG4TrackFastSimEval
  - provides info on tracks based on all detectors included in the Kalman filter
  - reconstructed and true info available
- CaloEvaluator
  - provides tower and cluster output for given calorimeter (central or forward)
  - object-based output makes event studies difficult
- JetEvaluator
  - provides reconstructed and true jets (position, energy, p_T, ...)

Standard eval tasks provide easy access to output for further processing. However, object-based output not suited for all studies!



One tree per object type (tracks, jets, calo, ...)

Evaluating your Output (2)

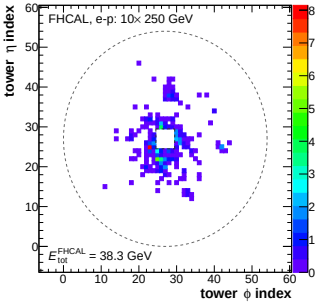


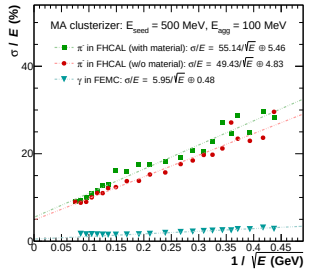
Event-based output with the EventEvaluator.cc/h

- Highly optimized event-based output for detailed studies
  - Hits, tracks, and track projections
  - Towers and clusters
  - Vertex
  - MC particles (primary and settable depth for secondaries)
- Allows for reclusterization, jet finding, tracking, and more on afterburner-level
- Can handle all currently implemented detector configurations!
- EventEvaluator is constantly evolving and new features and detectors added regularly!
- Default output for Fun4All_G4_FullDetectorModular.C
- Afterburner code for event tree available in AnalysisSoftwareEIC

EventEvaluator-like output is best suited for physics/performance studies!

IMHO: Event-based is much better than object-based.





Questions & Discussion

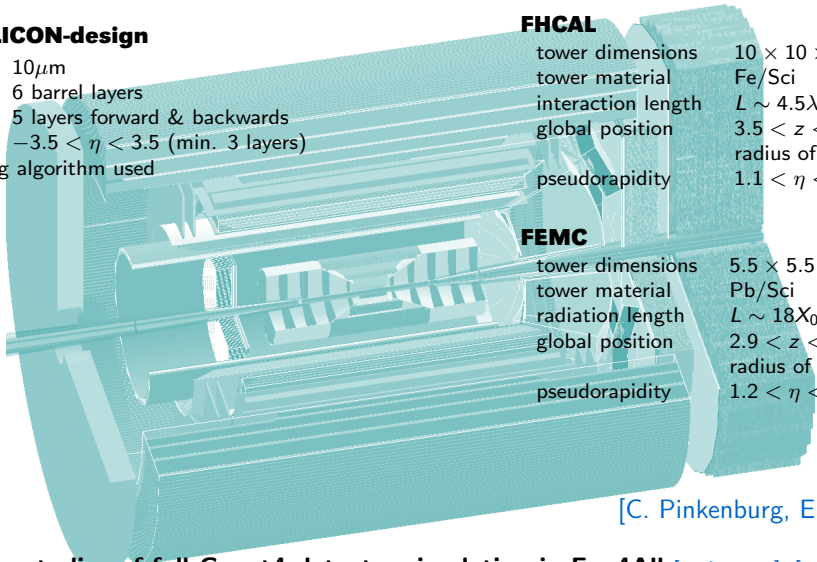
Feel free to also ask questions in Mattermost!

Backup

Baseline Detector Design for Studies

Tracker ALLSILICON-design

pitch	10μm
layers	6 barrel layers
	5 layers forward & backwards
pseudorapidity	$-3.5 < \eta < 3.5$ (min. 3 layers)
idealized tracking	algorithm used



FHCAL

tower dimensions	$10 \times 10 \times 81$ cm^3
tower material	Fe/Sci
interaction length	$L \sim 4.5\lambda$
global position	$3.5 < z < 4.5$m,
	radius of ~ 2.62m
pseudorapidity	$1.1 < \eta < 3.5$ (at z center)

FEMC

tower dimensions	$5.5 \times 5.5 \times 36.3$ cm^3
tower material	Pb/Sci
radiation length	$L \sim 18X_0$
global position	$2.9 < z < 3.3$m,
	radius of ~ 1.83m
pseudorapidity	$1.2 < \eta < 3.5$ (at z center)

[C. Pinkenburg, EIC simulation]

Performance studies of full Geant4 detector simulation in Fun4All [main repo], [post processing]

Baseline Detector Design for Studies

Tracker ALLSILICON-design

pitch	10μm
layers	6 barrel layers
	5 layers forward & backwards
pseudorapidity	$-3.5 < \eta < 3.5$ (min. 3 layers)
idealized tracking	algorithm used

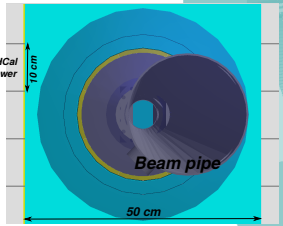
FHCAL

tower dimensions	$10 \times 10 \times 81$ cm^3
tower material	Fe/Sci
interaction length	$L \sim 4.5\lambda$
global position	$3.5 < z < 4.5$m,
	radius of ~ 2.62m
pseudorapidity	$1.1 < \eta < 3.5$ (at z center)

FEMC

tower dimensions	$5.5 \times 5.5 \times 36.3$ cm^3
tower material	Pb/Sci
radiation length	$L \sim 18X_0$
global position	$2.9 < z < 3.3$m,
	radius of ~ 1.83m
pseudorapidity	$1.2 < \eta < 3.5$ (at z center)

Forward beam pipe

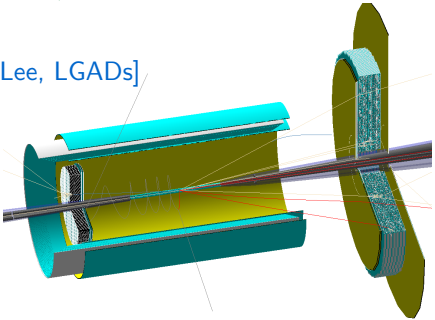


[C. Pinkenburg, EIC simulation]

Performance studies of full Geant4 detector simulation in Fun4All [main repo], [post processing]

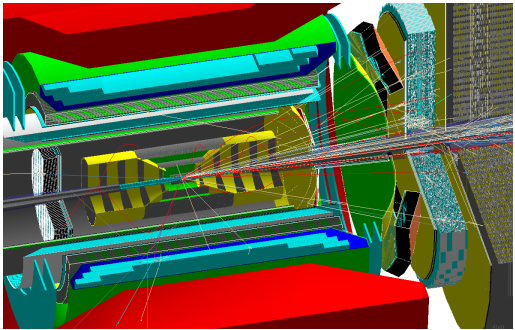
Detector modifications

[W. Lee, LGADs]



Timing Tracking Layer (TTL)

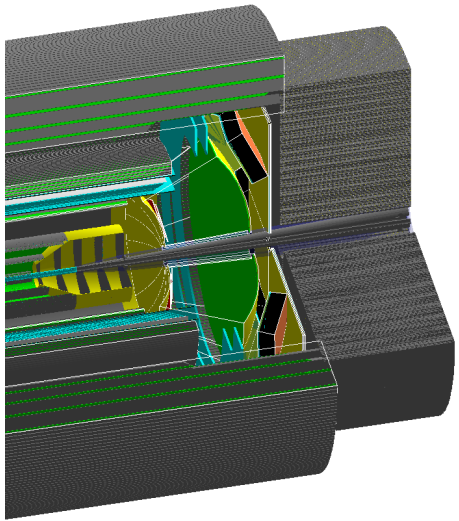
pitch	500μm
	$\sigma_{x,y} \sim 30\mu$m (AC-LGAD)
	$\sigma_{x,y} \sim 145\mu$m (std)
layers	2 barrel layers (1 after ECal)
	3 forward layers (1 after ECal)
	2 backward layers
pseudorapidity	$-3.7 < \eta < 3.7$
PID	$\sigma_t \sim 20$ps (single layer)



BEAST Magnet

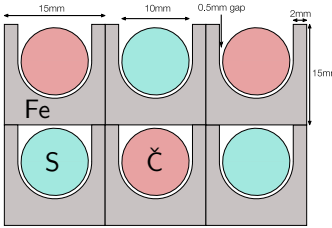
- $B = 3$ T
- superconducting
- outer HCal serves as flux return

Outlook: Alternative Calorimeter

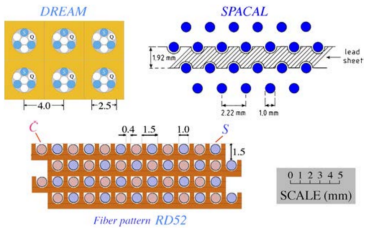


Dual Readout Calorimetry

- aiming for full detector simulations in Fun4All/Geant4
- exact DR design still to be chosen
  → e.g. IDEA concept replacing FEMC and FHCAL [link]
- advantage of small constant term in resolution
- simultaneous readout of electro-magnetic and hadronic response



Current Implementation

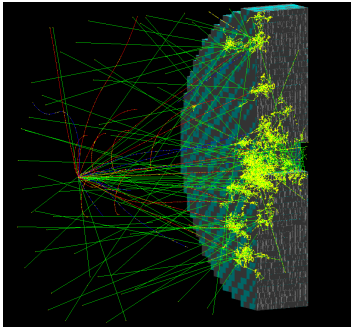


[R. Ferrari, Dual Readout Calorimetry]

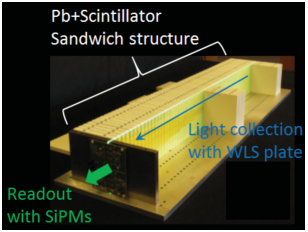
Current HCAL Design

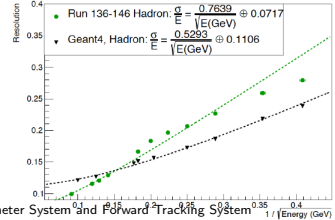
Design based on STAR forward upgrade

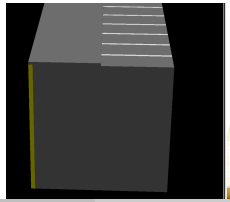
tower dimensions	$10 \times 10 \times 100$ cm^3 (2044 towers in total)
tower material	44 layers: 20mm Fe and 2.3mm plastic Sci
interaction length	$L \sim 4.5\lambda$
pseudorapidity	$1.07 < \eta < 3.47$
	tower $\Delta\eta = 0.04$ at $\eta = 1.1$
	tower $\Delta\eta = 0.35$ at $\eta = 3.0$
global position	$3.5 < z < 4.5$m
	radius of ~ 2.62m
readout	avg of 8 SiPM per WLS plate

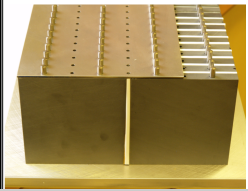


Simulation of an e–p collision in Fun4All with FHCAL







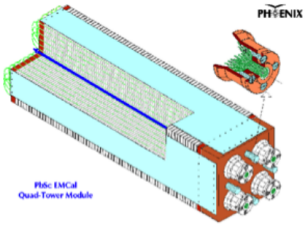


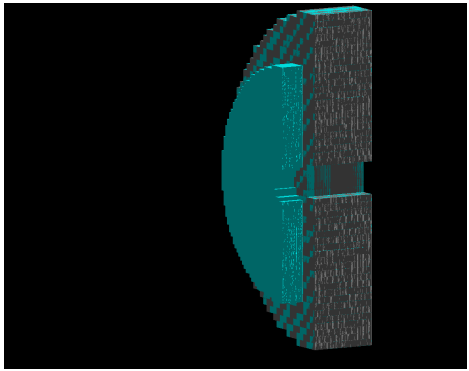
HCAL information from: The STAR Forward Calorimeter System and Forward Tracking System

Current ECAL Design

Design based on PHENIX calorimeter

tower dimensions	$5.5 \times 5.5 \times 37.5$ cm^3 (3244 towers in total)
tower material	Pb-Sci: 60 layers
interaction length	$L \sim 18X_0$
pseudorapidity	$1.18 < \eta < 3.47$
global position	center at 3.1m from IP
	radius of ~ 1.8m
readout	via SiPM
sampling fraction	29.5%
expected resolution	$8.1\%/\sqrt{E} \oplus 2.1$

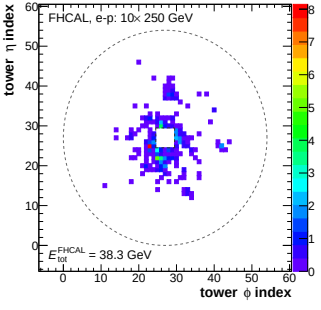


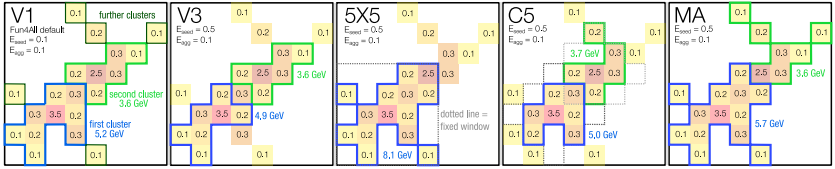


Forward ECAL and HCAL in Fun4All

ECAL information from: An EIC Detector Built Around The sPHENIX Solenoid, NIM A499 (2003) 521-536

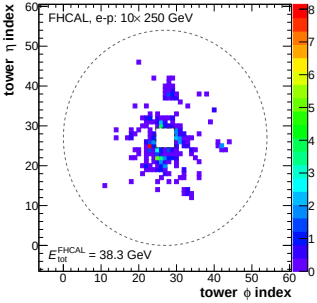
Clusterization performance

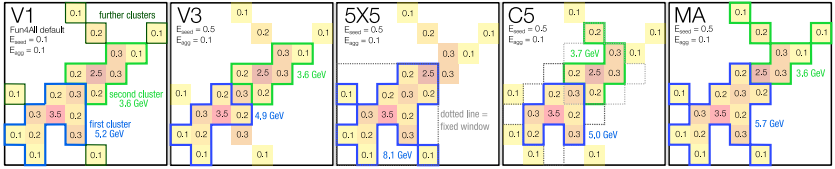




- Cluster finding and association to single particle challenging task in forward direction
- Implemented different algorithms:
  - Fun4All default w/o thresholds (V1)
  - simple splitting w/ thresholds (V3)
  - areas based: square (NxN) or circle (CN)
  - splitting including diagonals w/ thr. (MA)
- Different cluster finding efficiencies, resolutions & other properties

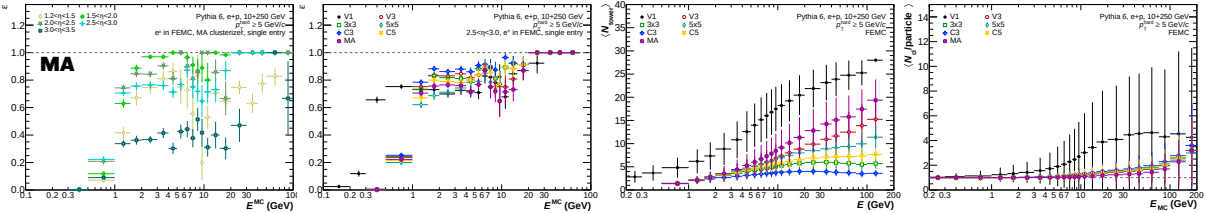
Clusterization performance



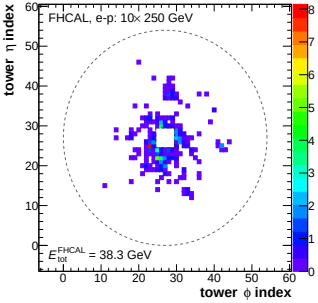


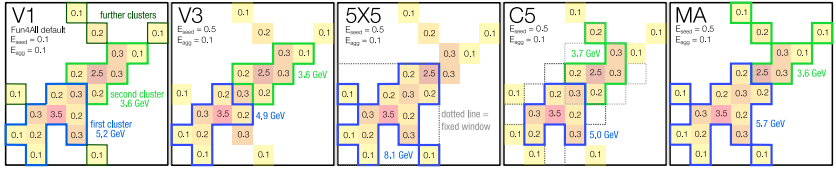
- Cluster finding and association to single particle challenging task in forward direction
- Implemented different algorithms:
  - Fun4All default w/o thresholds (V1)
  - simple splitting w/ thresholds (V3)
  - areas based: square (NxN) or circle (CN)
  - splitting including diagonals w/ thr. (MA)
- Different cluster finding efficiencies, resolutions & other properties

FEMC



Clusterization performance





- Cluster finding and association to single particle challenging task in forward direction
- Implemented different algorithms:
  - Fun4All default w/o thresholds (V1)
  - simple splitting w/ thresholds (V3)
  - areas based: square (NxN) or circle (CN)
  - splitting including diagonals w/ thr. (MA)
- Different cluster finding efficiencies, resolutions & other properties

FHCAL