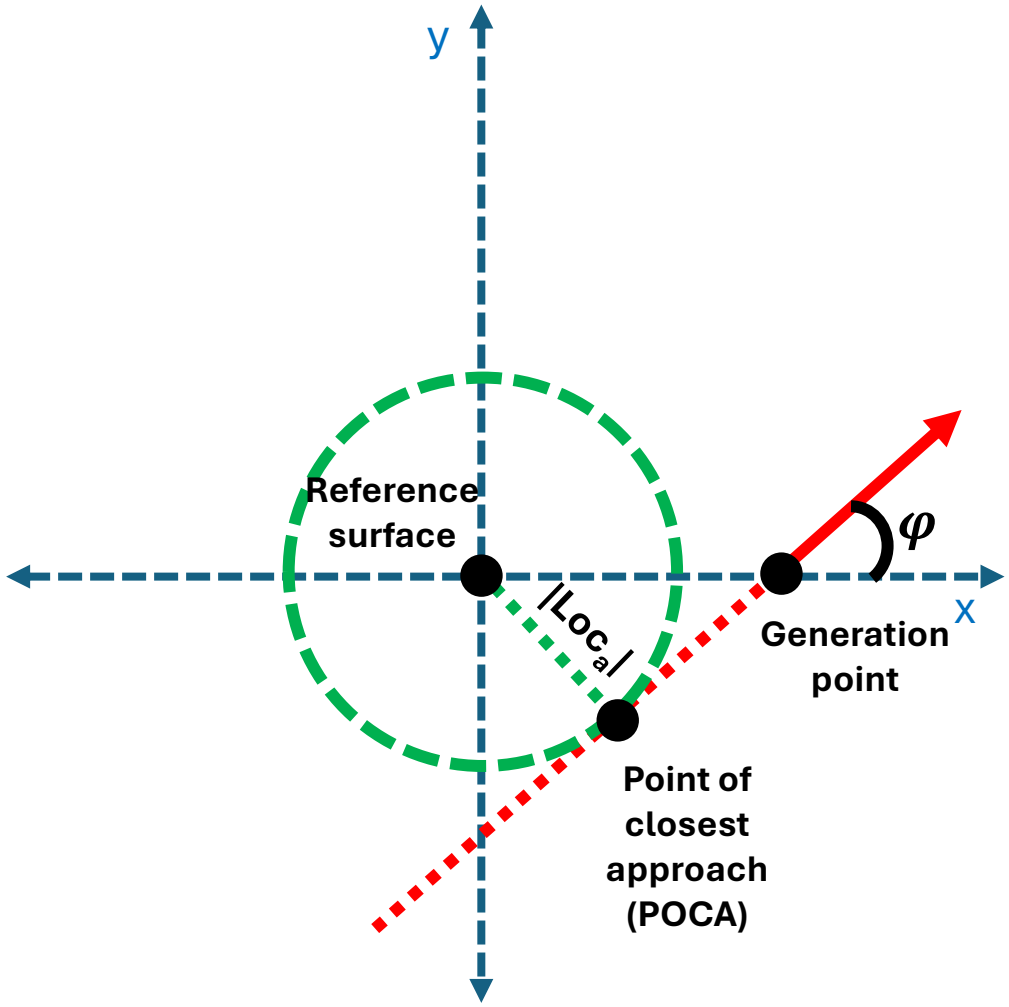
Calculating track parameters at primary vertex

Barak Schmookler

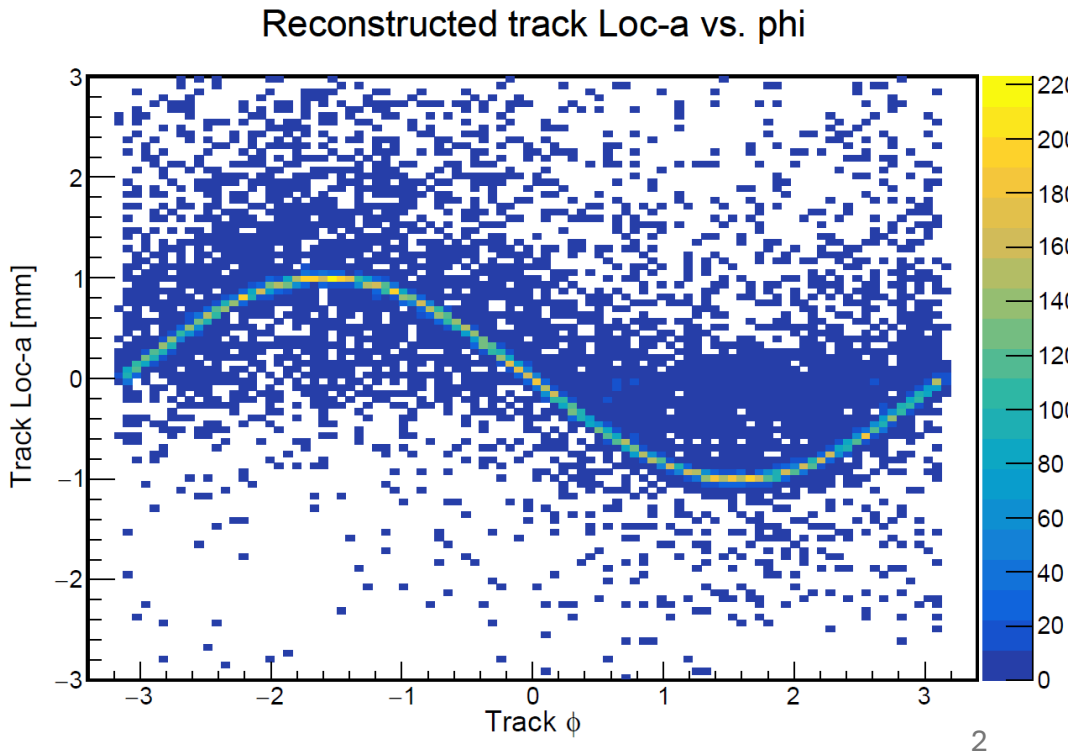
Reconstructed track parameters



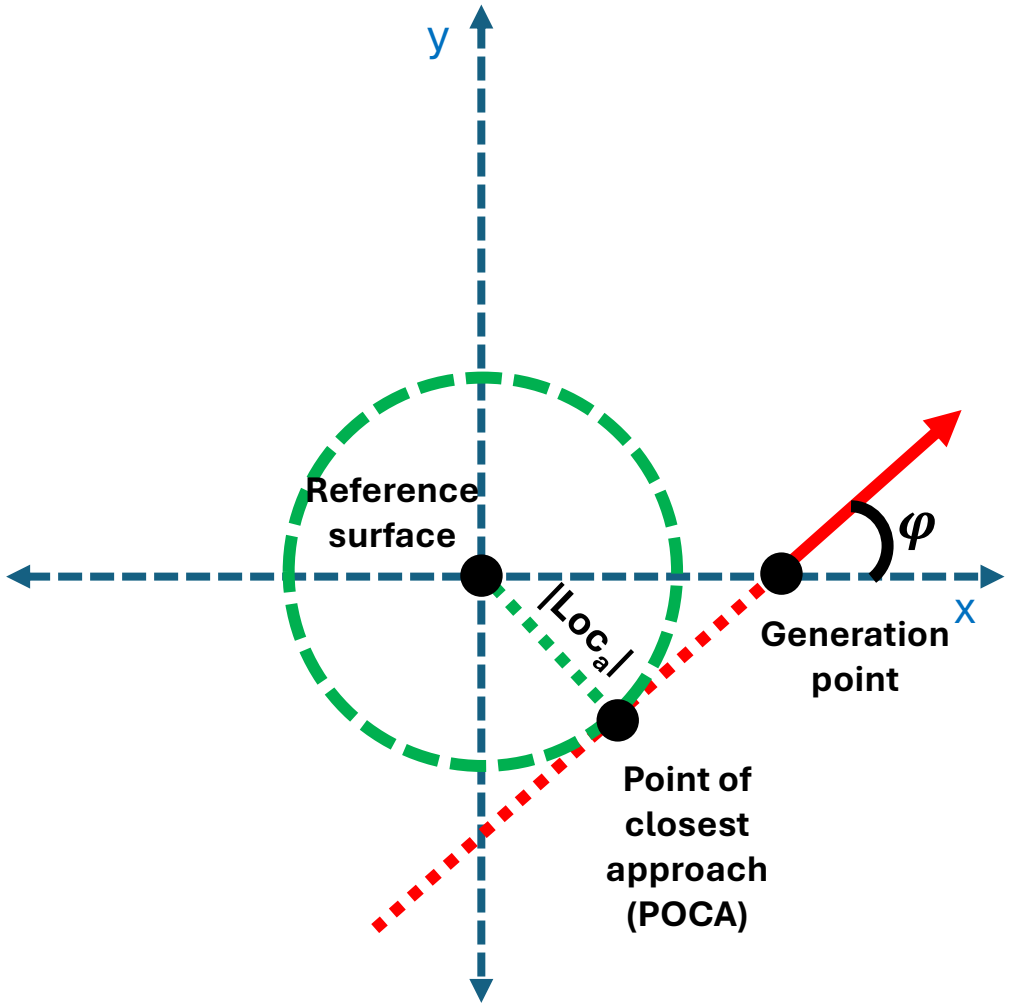
Following the track fit, we choose to save the parameters for a given track at the point of closest approach to the beamline (z-axis).

This is accomplished using the Acts PerigeeSurface class.

(vx, vy, vz) = (+1,0,0) mm

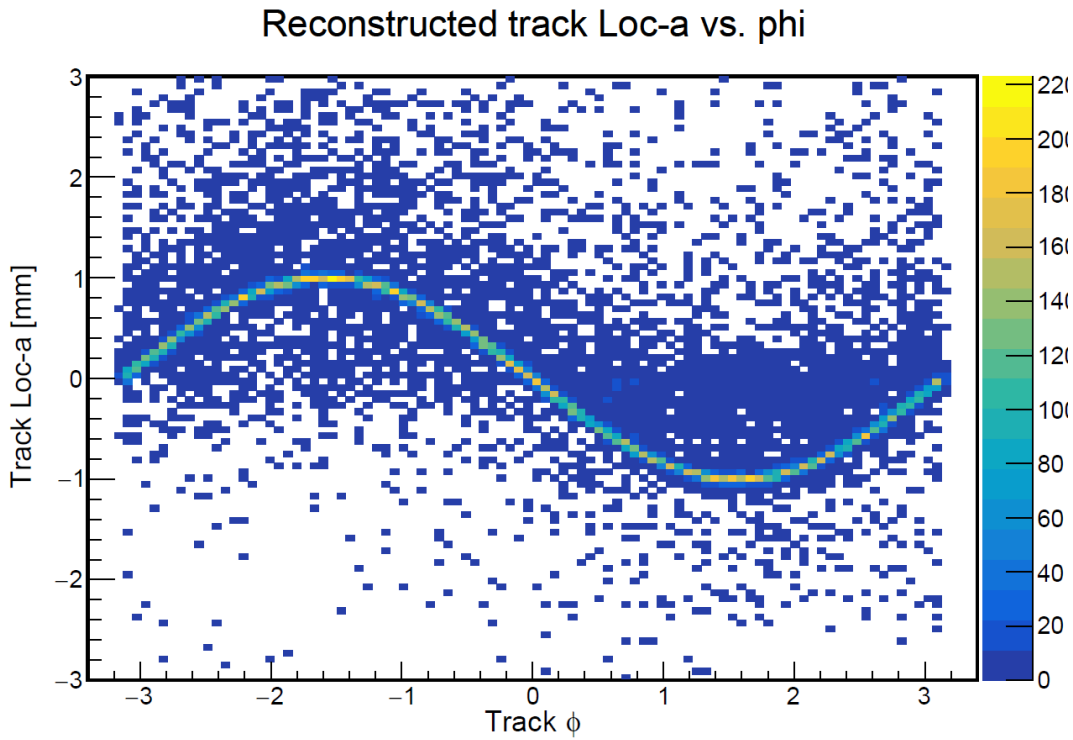


Reconstructed track parameters

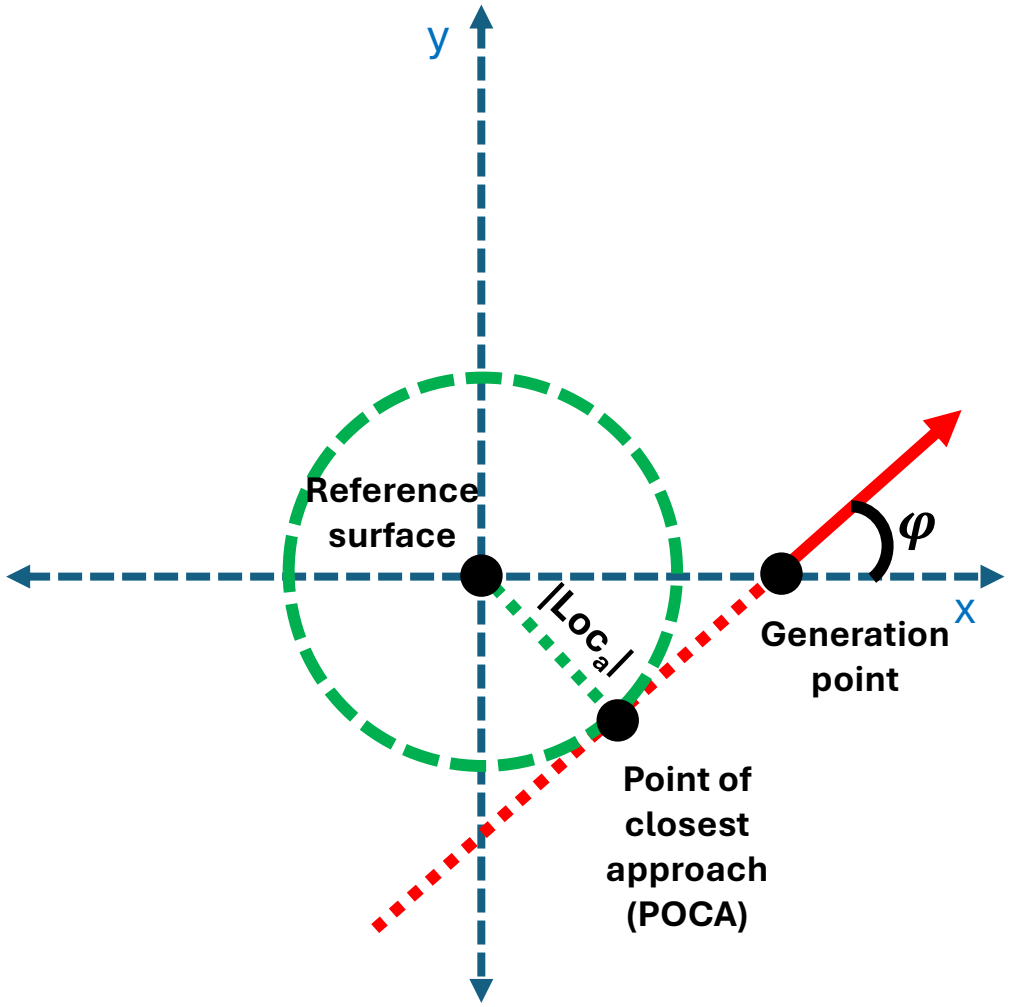


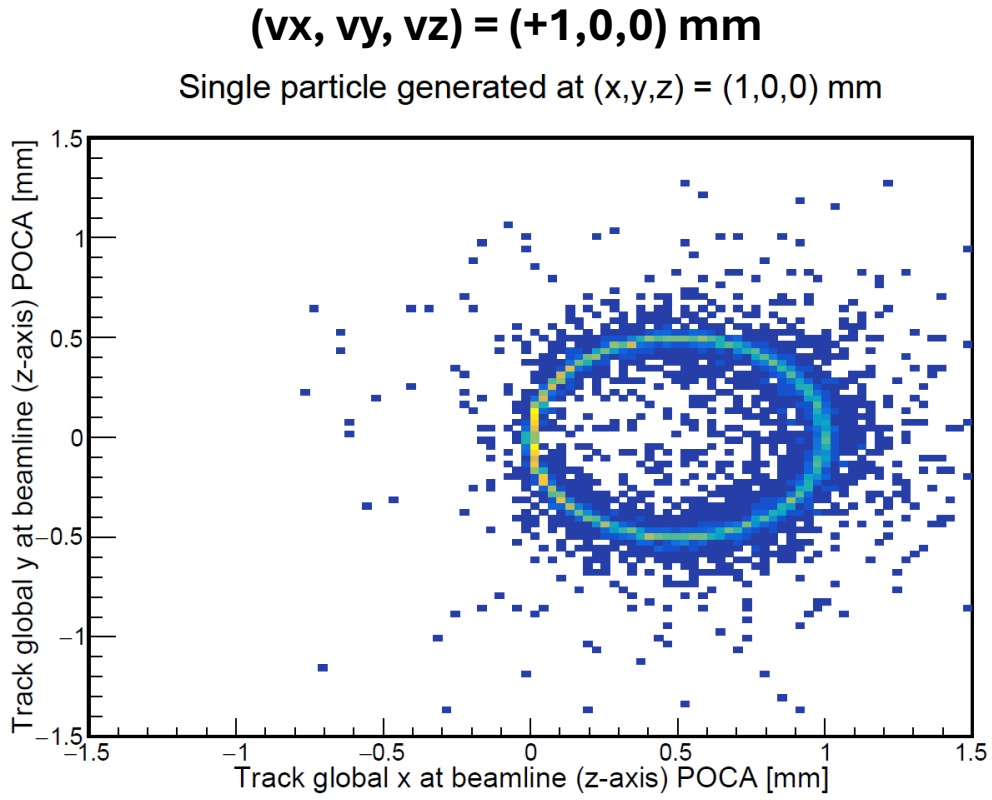
This is the track's position in local coordinates. We can convert to global coordinates using the **PerigeeSurface::localtoglobal()** method.

(vx, vy, vz) = (+1,0,0) mm



Reconstructed track parameters





Track parameters at primary vertex

- After reconstructing the tracks – and saving the track parameters at the beamline POCA – we run the vertex finder/fitter to get the primary vertex position.
- We now want to determine the track parameters at the 3D DCA point w.r.t. the found primary vertex.
- We can use the Acts class called the *ImpactPointEstimator* to do this.
- Using this class in an analysis script, however, requires some effort to load the DD4Hep and Acts information correctly.

Class Acts::ImpactPointEstimator

Result<BoundTrackParameters> **estimate3DImpactParameters**(*const* GeometryContext &gctx, *const* Acts::MagneticFieldContext &mctx, *const* BoundTrackParameters &trkParams, *const* Vector3 &vtxPos, State &state) *const*

Creates track parameters bound to plane at point of closest approach in 3d to given reference position.

The parameters and errors are defined on the plane intersecting the track at point of closest approach, with track orthogonal to the plane and center of the plane defined as the given reference point (vertex).

Parameters:
- **gctx** – The geometry context
- **mctx** – The magnetic field context
- **trkParams** – Track parameters
- **vtxPos** – Reference position (vertex)
- **state** – The state object

Returns: New track params

How to do this in an analysis script…

```
// Load DD4Hep geometry
dd4hep::Detector& detector = dd4hep::Detector::getInstance();
detector.fromCompact("/opt/detector/epic-main/share/epic/epic_craterlake.xml");
dd4hep::DetElement geometry = detector.world();

// Convert DD4Hep geometry to tracking geometry
Acts::GeometryContext trackingGeoCtx;
auto logger = Acts::getDefaultLogger("DD4hepConversion", Acts::Logging::Level::INFO);
Acts::BinningType bTypePhi = Acts::equidistant;
Acts::BinningType bTypeR = Acts::equidistant;
Acts::BinningType bTypeZ = Acts::equidistant;
double layerEnvelopeR = Acts::UnitConstants::mm;
double layerEnvelopeZ = Acts::UnitConstants::mm;
double defaultLayerThickness = Acts::UnitConstants::fm;
using Acts::sortDetElementsByID;

std::shared_ptr<const Acts::TrackingGeometry> trackingGeometry{nullptr};
trackingGeometry = Acts::convertDD4hepDetector(geometry,*logger,bTypePhi,bTypeR,bTypeZ,layerEnvelopeR,layerEnvelopeZ,defaultLayerThick

// Define Perigee surface at which reconstructed track parameters are set
auto perigee = Acts::Surface::makeShared<Acts::PerigeeSurface>(Acts::Vector3(0,0,0));

// Get Magnetic field context
Acts::MagneticFieldContext fieldctx;
std::shared_ptr<const Acts::DD4hepFieldAdapter> field_provider = std::make_shared<const Acts::DD4hepFieldAdapter>(detector.field());
Acts::MagneticFieldProvider::Cache field_cache = field_provider->makeCache(fieldctx);
```

How to do this in an analysis script...

```
// Stepper and Propagator
using Stepper    = Acts::EigenStepper<>;
using Propagator = Acts::Propagator<Stepper>;

Stepper stepper(field_provider);
Propagator propagator(stepper);

// Create Impact Point Estimator
Acts::ImpactPointEstimator::Config ImPoEs_cfg(field_provider,std::make_shared<Propagator>(propagator));

Acts::ImpactPointEstimator::State ImPoEs_state;
ImPoEs_state.fieldCache = field_cache;

Acts::ImpactPointEstimator ImPoEs(ImPoEs_cfg);

// Create 'vertex' at particle's creation point -- which is (x,y,z) = (1,0,0) mm
Acts::Vector3 vtx_pos(1.0 * Acts::UnitConstants::mm, 0, 0);
```

```
//---- Part 2: Get track parameters at 3D DCA to creation point at (x,y,z) = (1,0,0) mm ----
auto result = ImPoEs.estimate3DImpactParameters(trackingGeoCtx,fieldctx,track_parameters,vtx_pos,ImPoEs_state);

if(result.ok()){
    Acts::BoundTrackParameters trk_boundpar_vtx = result.value();
    const auto& trk_vtx_params  = trk_boundpar_vtx.parameters();

    h2->Fill(trk_vtx_params[Acts::eBoundLoc0]);
}
```

Code has been uploaded to ‘snippets’ repository

main snippets / Tracking / ImpactPointEstimator / ↑ Top

..

README.md	Added ImpactPointEstimator analysis	4 minutes ago
pca_global_impactpoint.C	Added ImpactPointEstimator analysis	4 minutes ago

README.md

Simple analysis code using ImpactPointEstimator

Generating events from (x,y,z) = (1,0,0)mm

In the container, do the following:

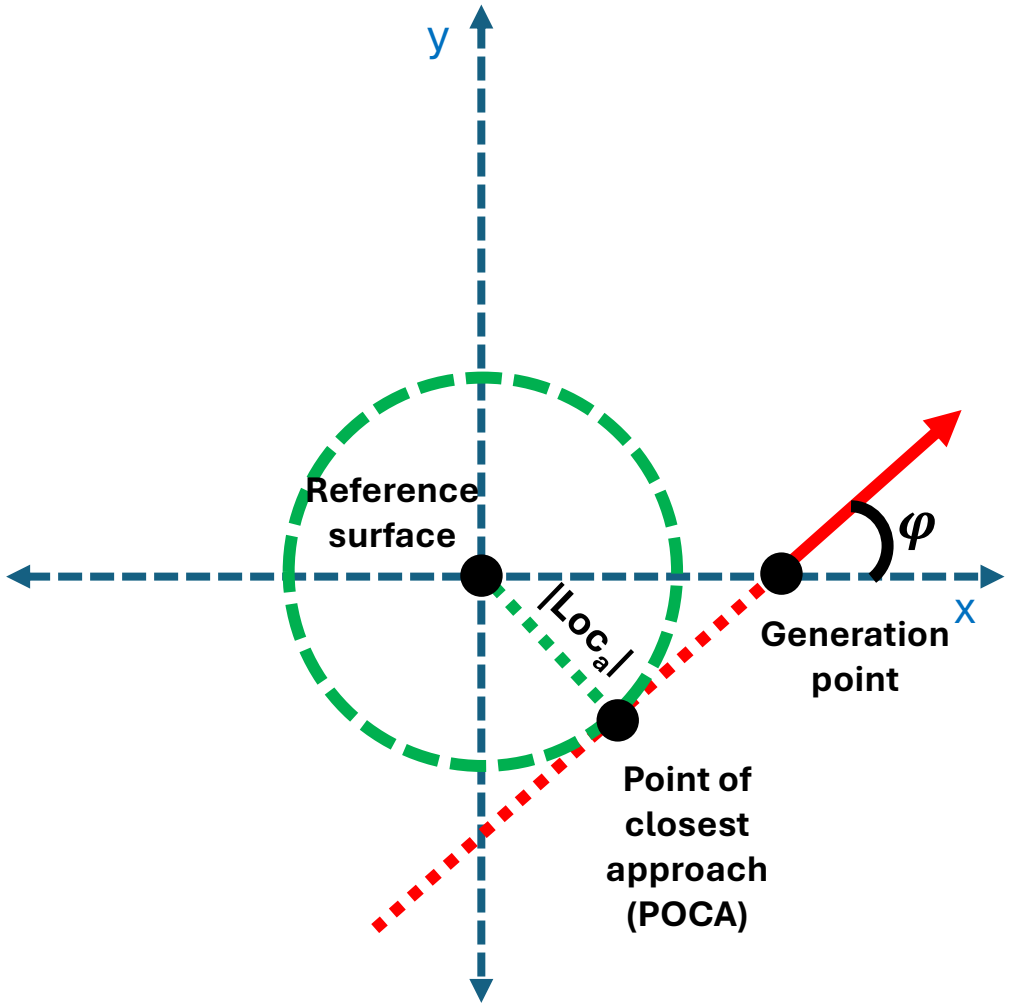
```
mkdir output
mkdir output/log

source /opt/detector/epic-main/bin/thisepic.sh

npsim --compactFile $DETECTOR_PATH/epic_craterlake.xml --enableGun --gun.distribution 'eta' \
--gun.thetaMax 3.106 --gun.thetaMin 0.036 --gun.momentumMin "0.5*GeV" --gun.momentumMax "20*GeV" \
--numberOfEvents 10000 --gun.position 1,0,0 --outputFile output/output_1_0_0.edm4hep.root

eicrecon -Ppodio:output_file=output/eicrecon_out_1_0_0.root -Pjana:nevents=10000 -Pdd4hep:xml_files=epic_craterlake.xml output/
```

Running the analysis code

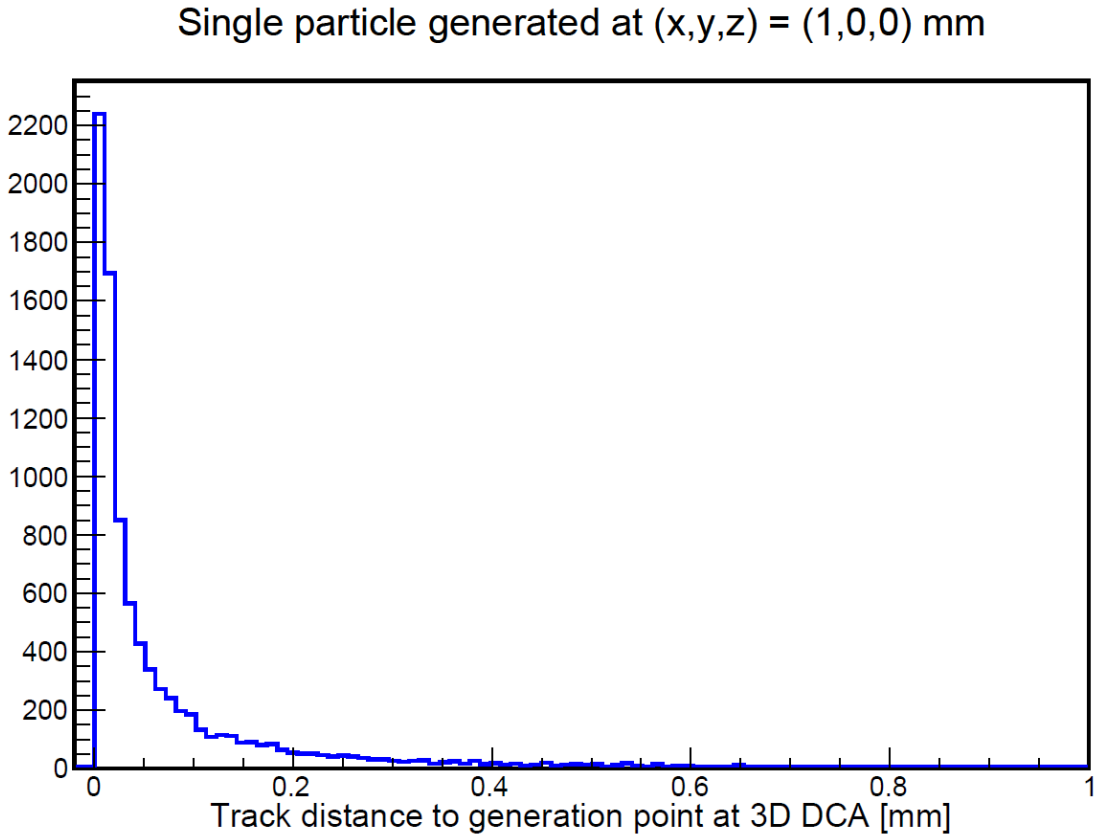
The analysis code needs to be run in the container.

```
mkdir plots
source /opt/detector/epic-main/bin/thisepic.sh
root -l -b -q pca_global_impactpoint.C
```

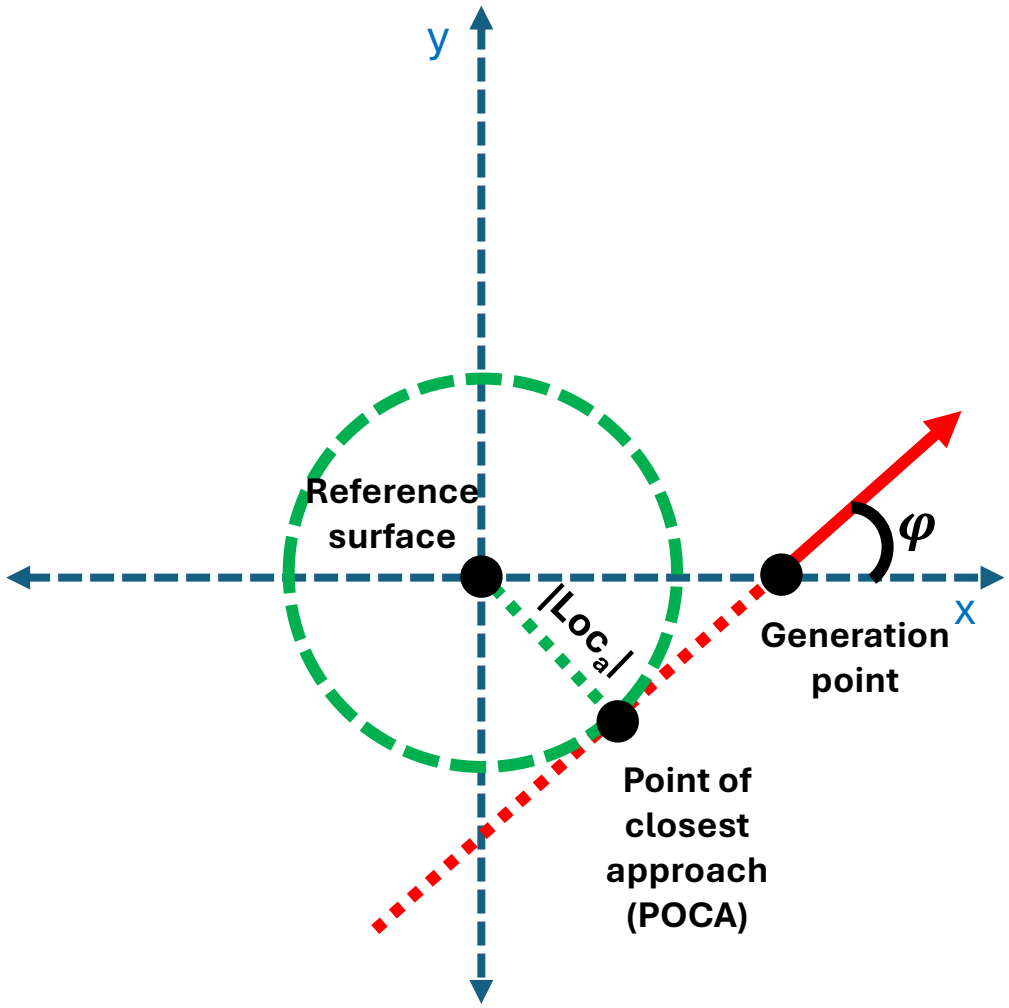
Use the ImpactPointEstimator to calculate the closest distance to (x,y,z) = (1,0,0) mm

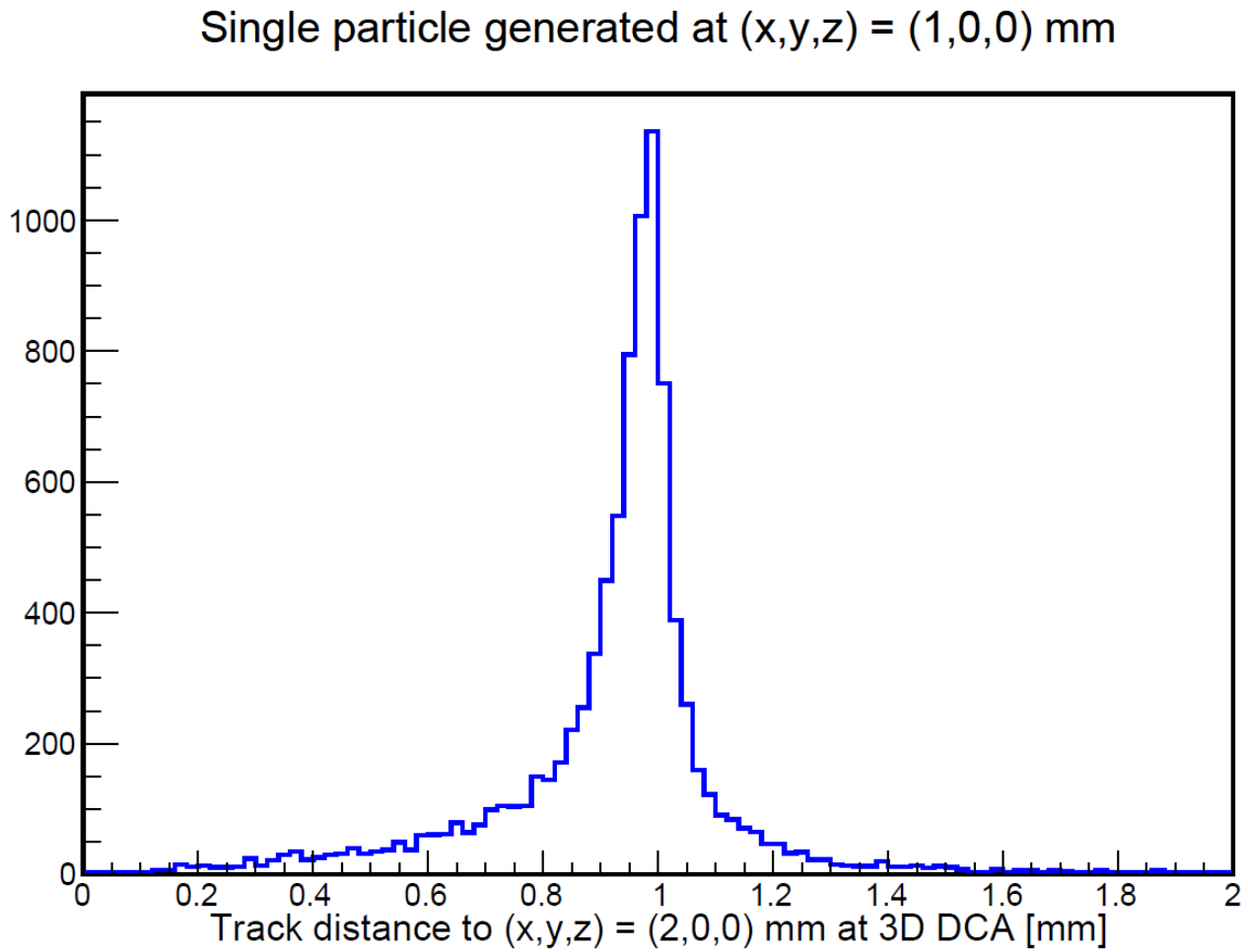


Since the particles are thrown from (vx, vy, vz) = (+1,0,0) mm, we expect this distance to be small.

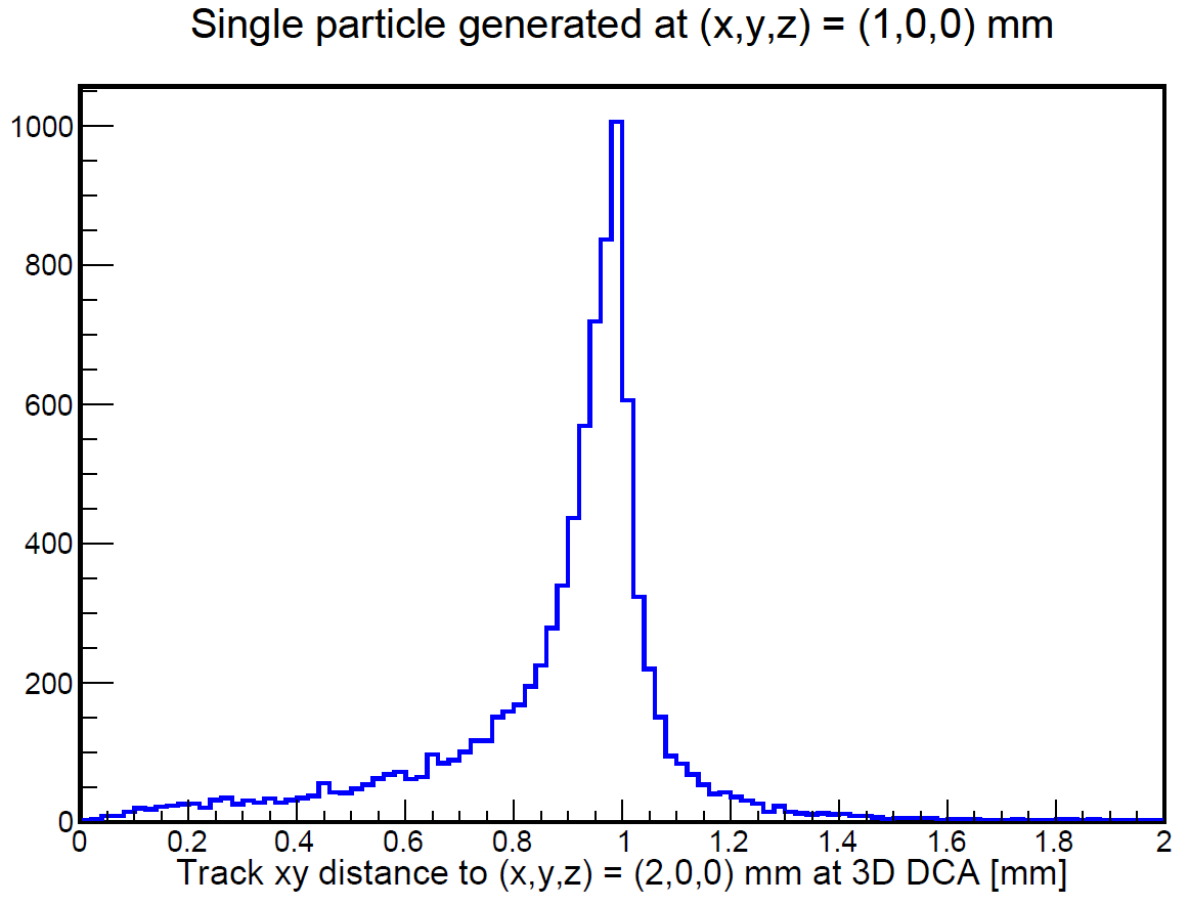


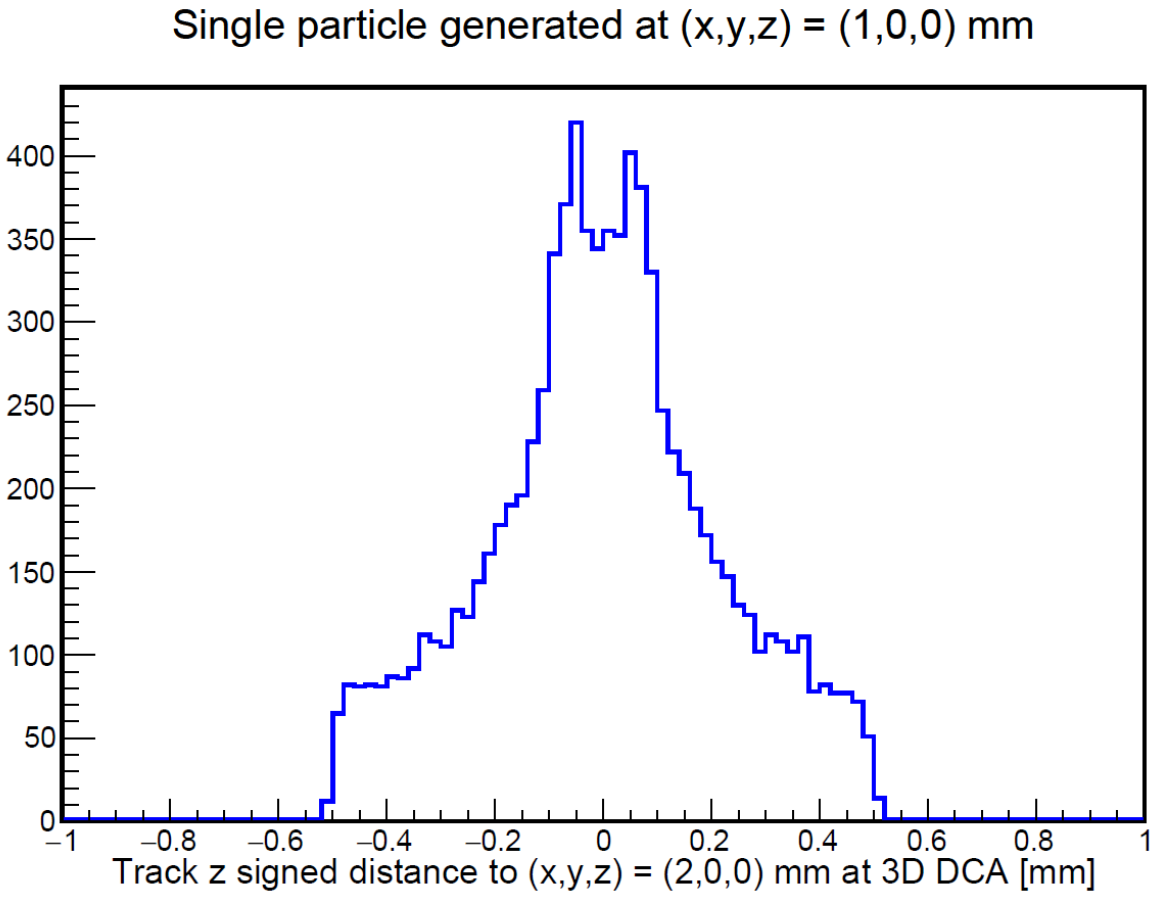
What if we calculate the closest distance to (x,y,z) = (2,0,0) mm?





Converting to global coordinates allows us to calculate DCA_{xy} and DCA_z

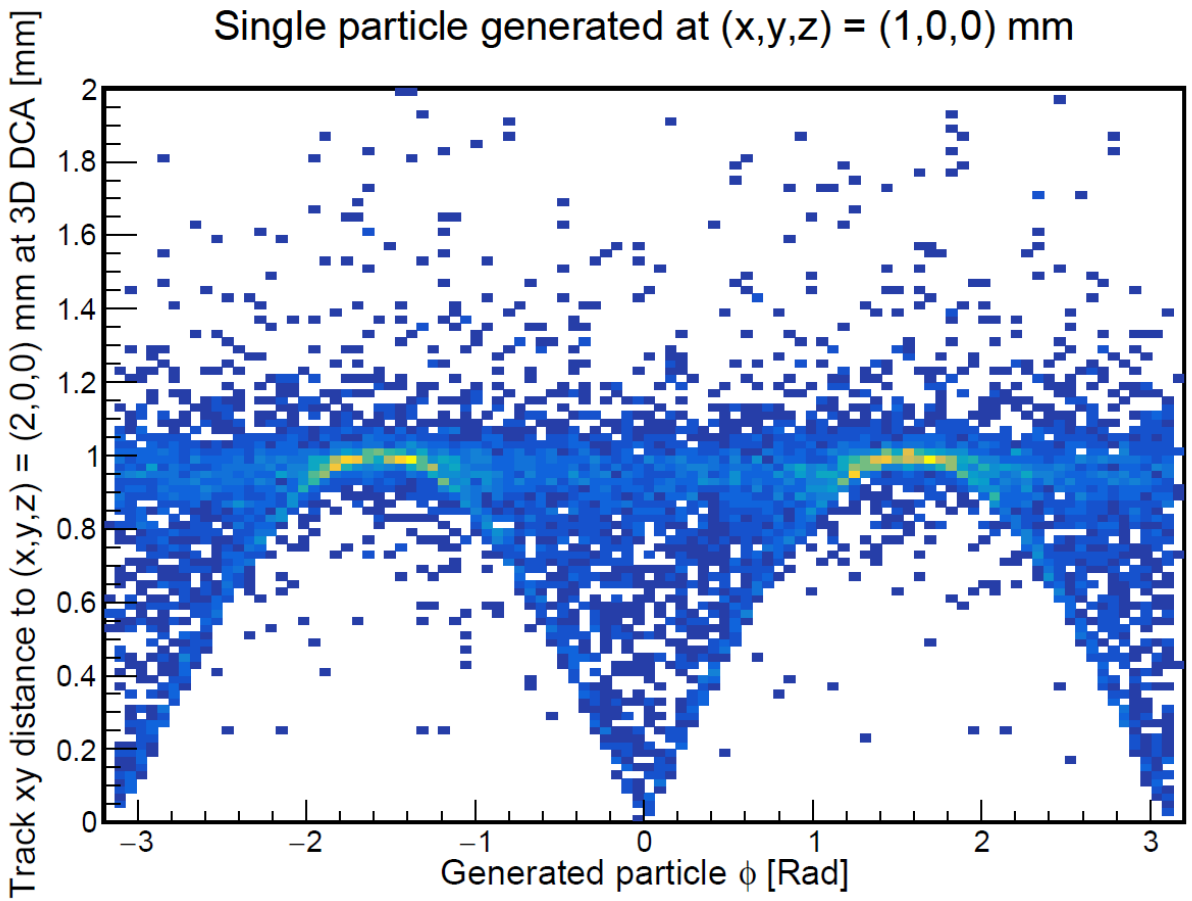


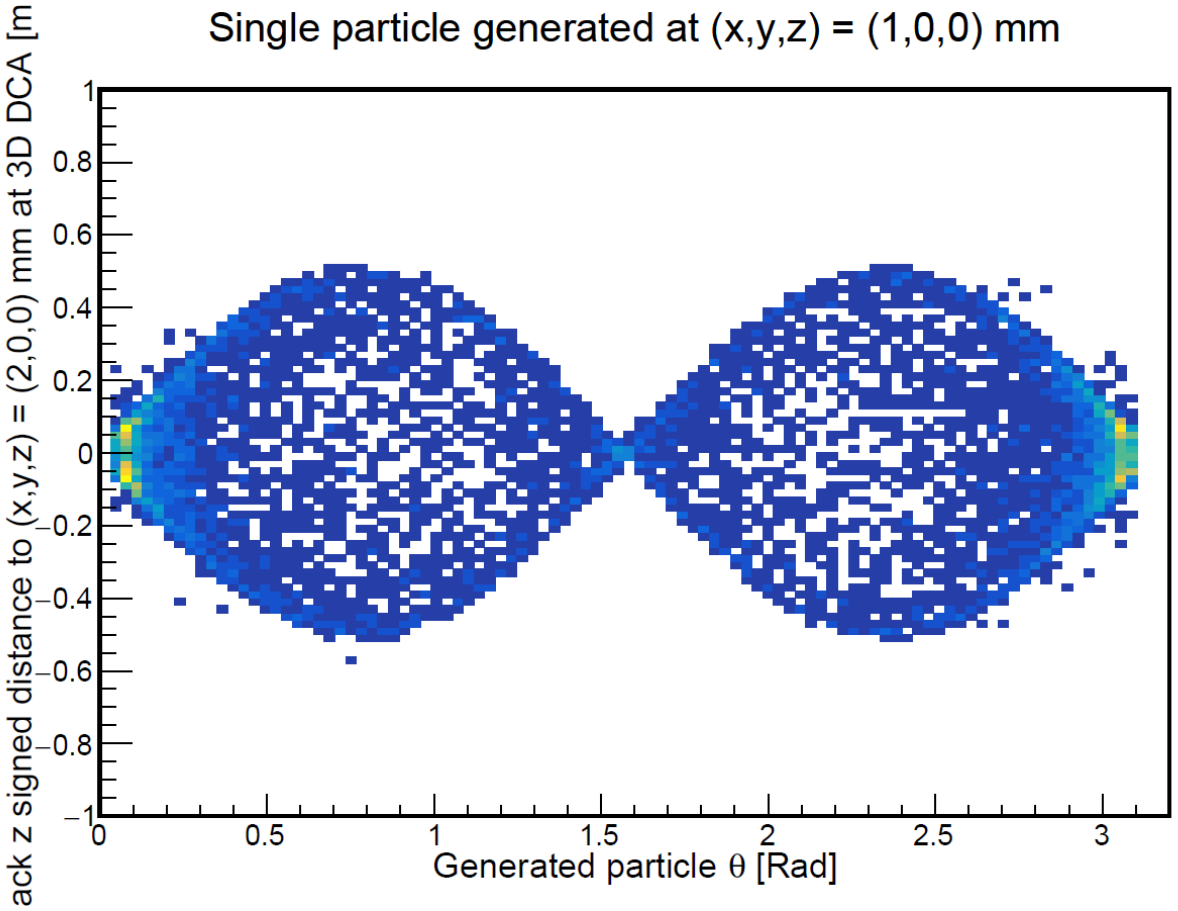


$$DCA_{xy} = \sqrt{(Trk_x - 2)^2 + (Trk_y - 0)^2}\ \text{[mm]}$$

$$DCA_z = Trk_z - 0\ \text{[mm]}$$

Converting to global coordinates allows us to calculate DCA_{xy} and DCA_z

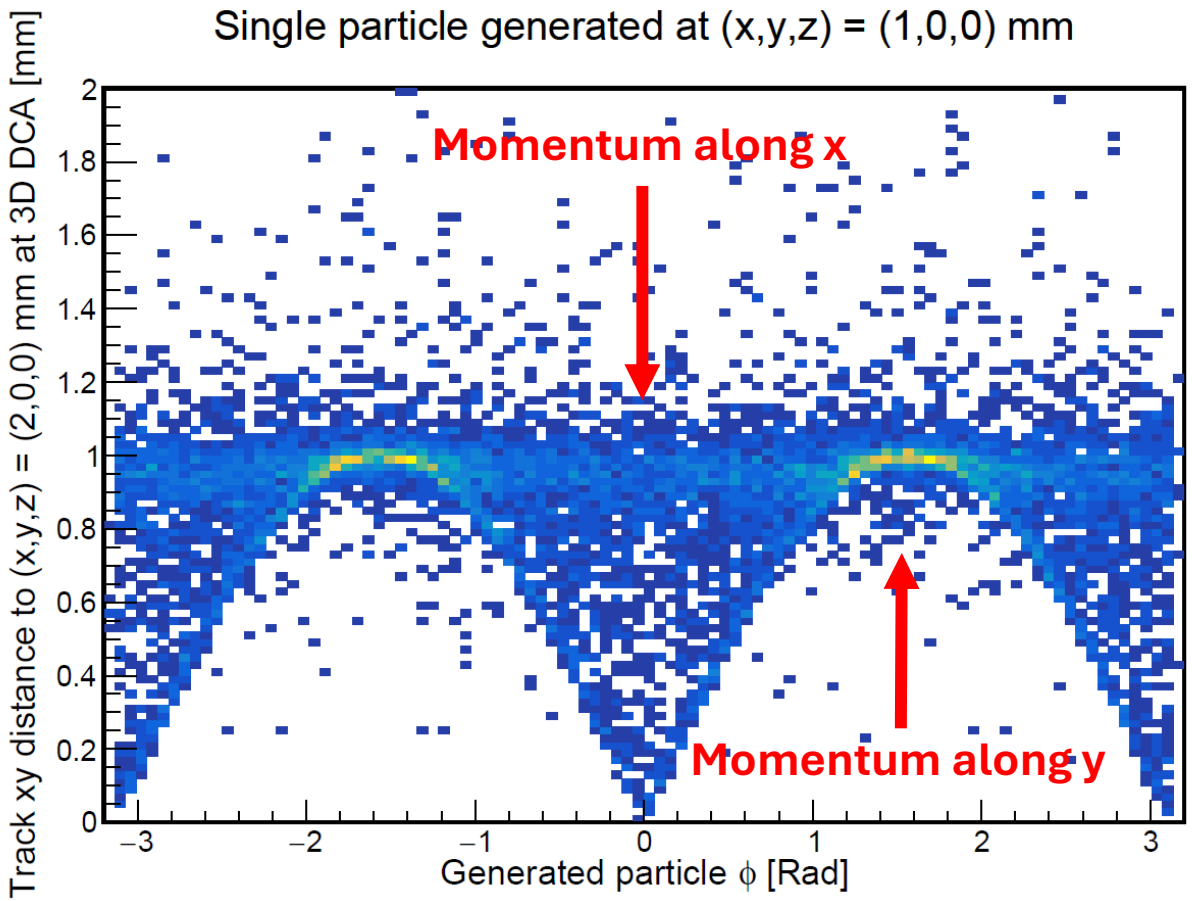


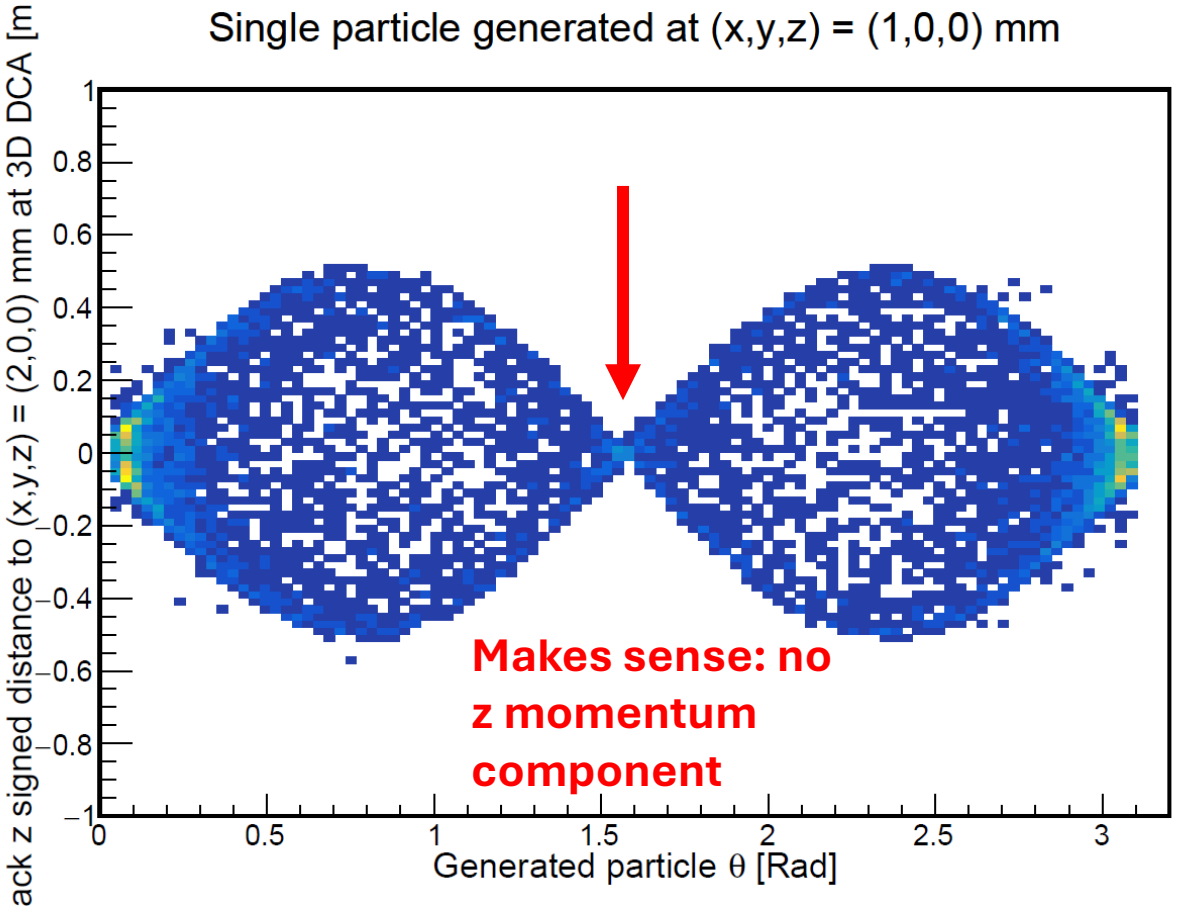


$$DCA_{xy} = \sqrt{(Trk_x - 2)^2 + (Trk_y - 0)^2}\ \text{[mm]}$$

$DCA_z = Trk_z - 0$ [mm]

Converting to global coordinates allows us to calculate DCA_{xy} and DCA_z





$$DCA_{xy} = \sqrt{(Trk_x - 2)^2 + (Trk_y - 0)^2}\ \text{[mm]}$$

$DCA_z = Trk_z - 0$ [mm]

Next steps: applying this to DIS events

- We can now extract the track parameters at the primary vertex. This includes the track DCA to the primary vertex.
- We can go on to apply this to DIS events (e.g. for D0 mass peak). See work by Rongrong at last Jets/HF WG meeting.
- One thing is missing – calculating the DCA between any two reconstructed tracks