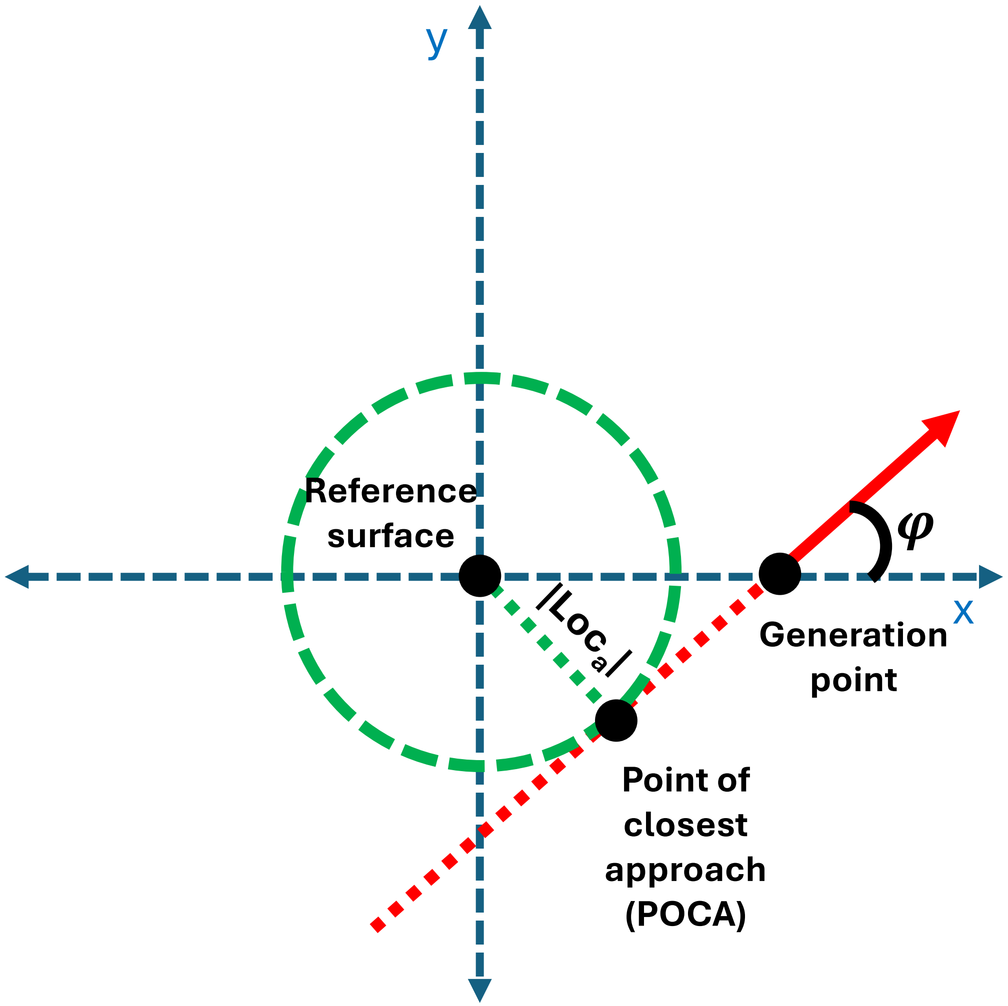
Calculating track parameters w.r.t. primary vertex

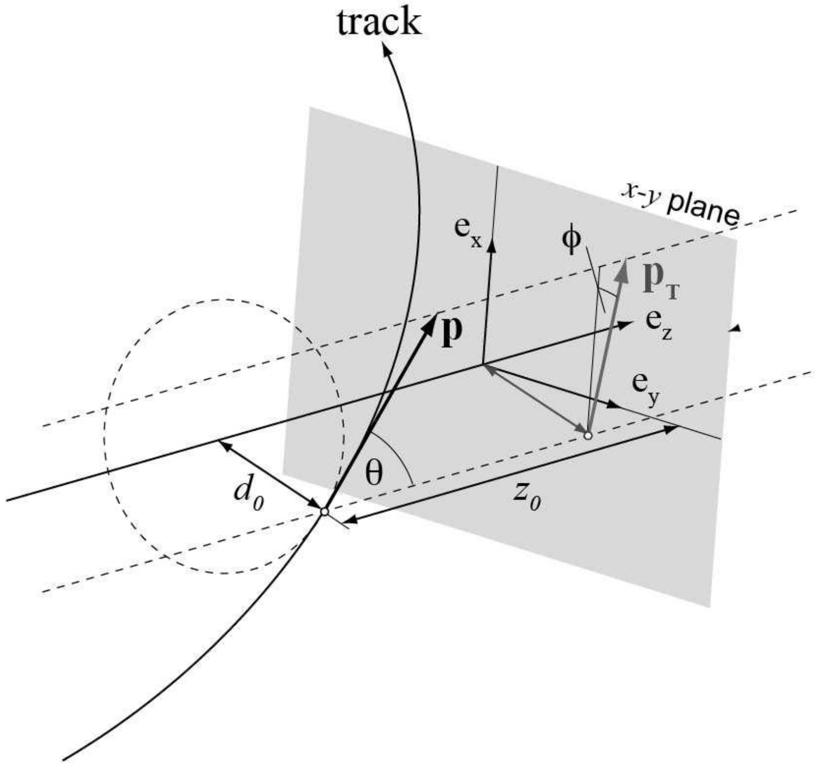
Barak Schmookler

Reconstructed track parameters

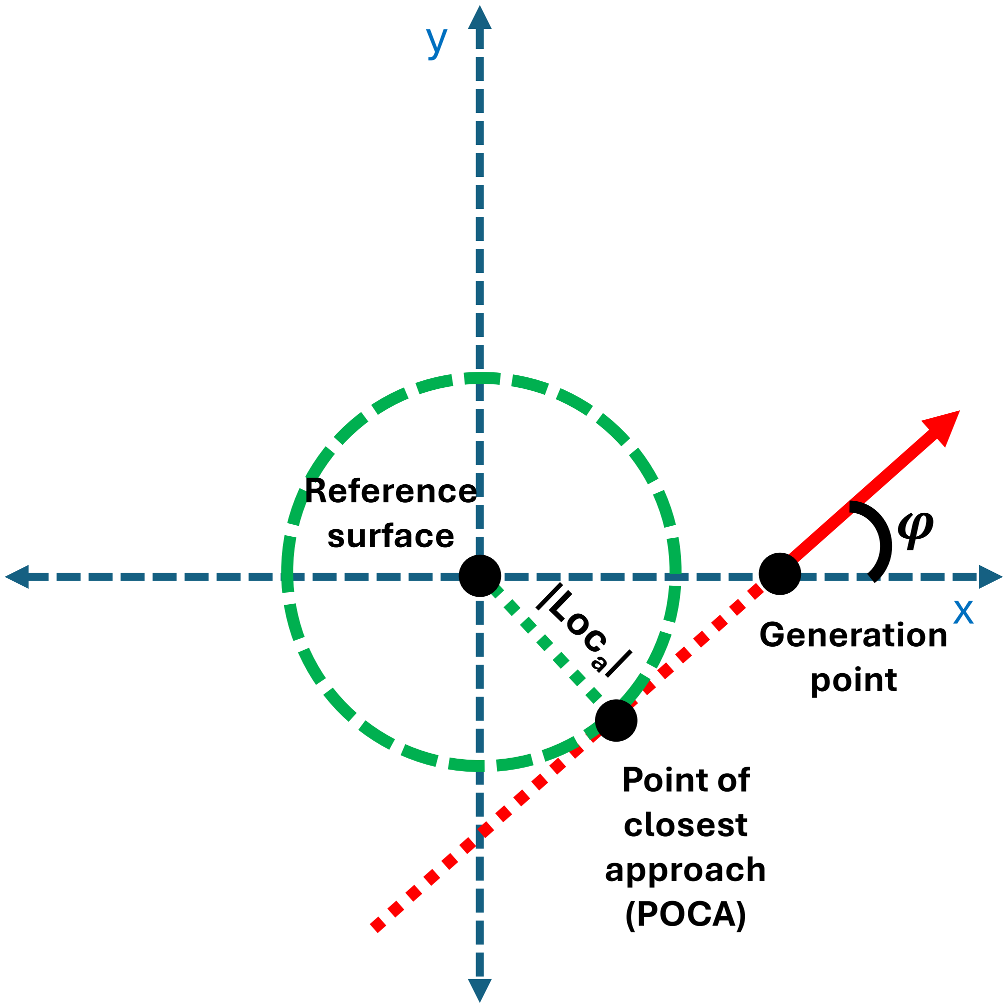


Following the track fit, we choose to save the parameters for a given track at the point of closest approach to the beamline (z-axis).

This is accomplished using the Acts *PerigeeSurface* class.

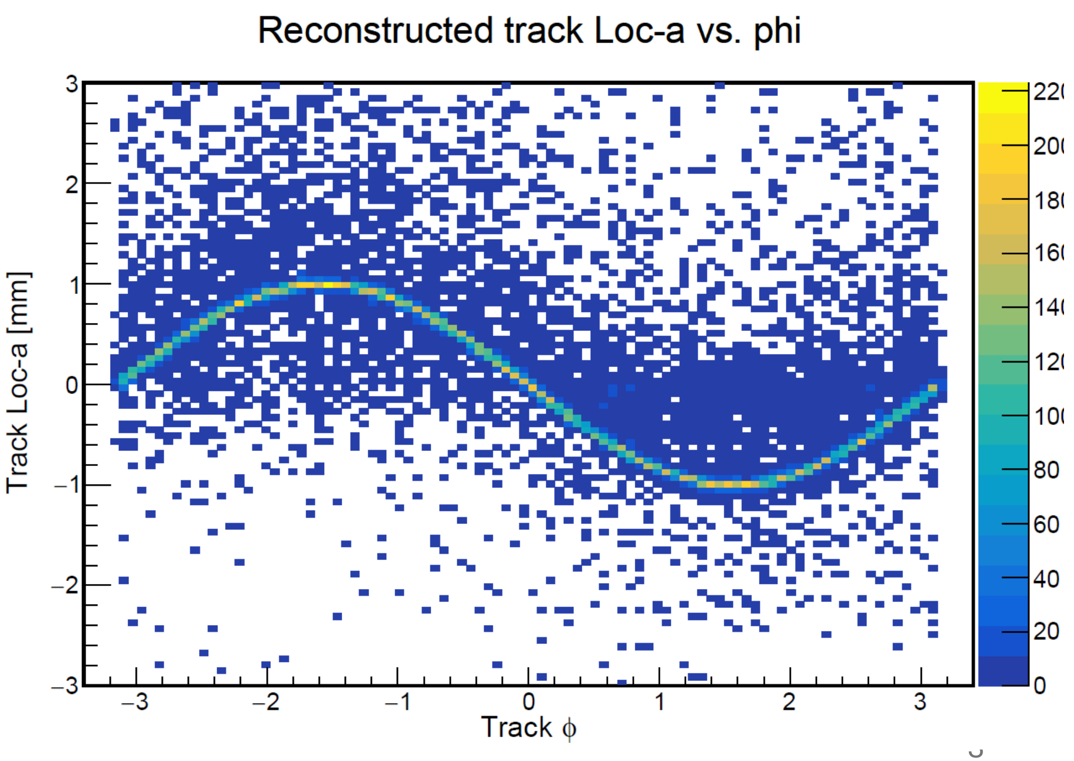


Reconstructed track parameters

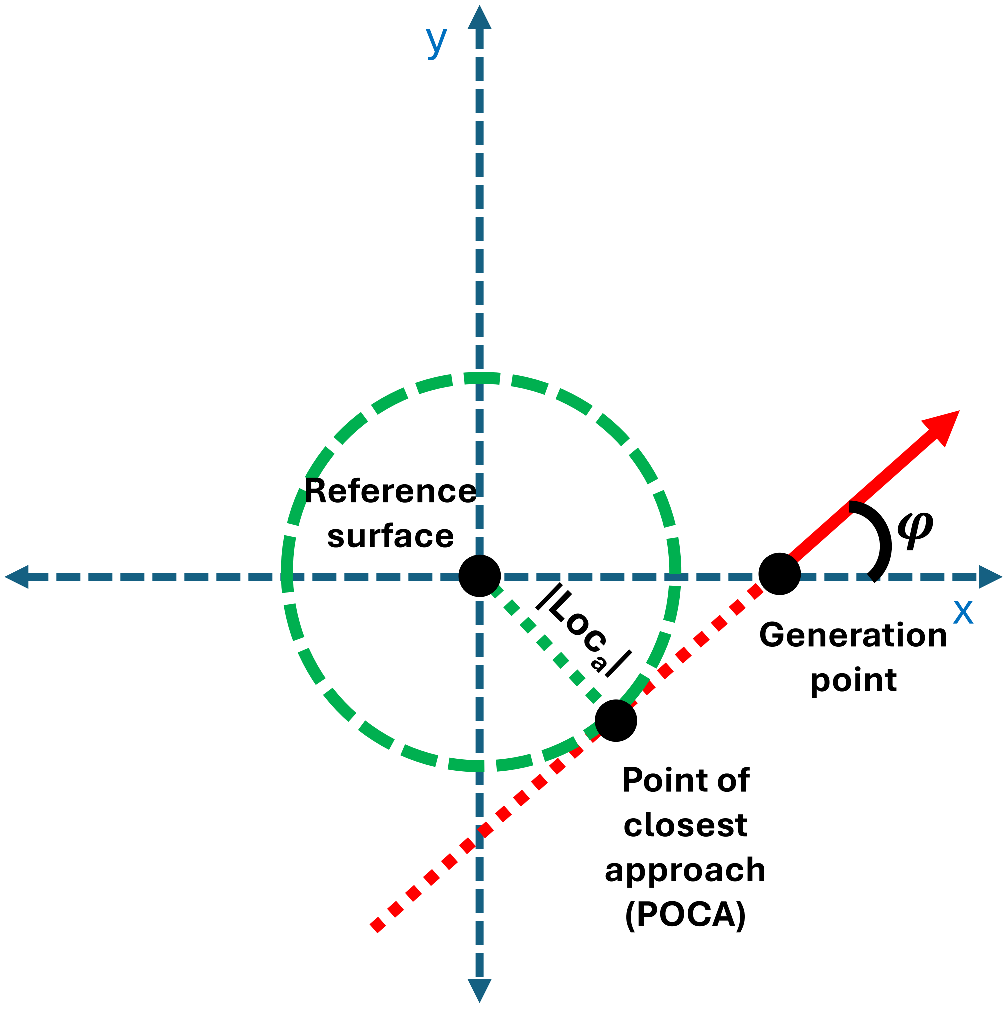


This is the track's position in local coordinates. We can convert to global coordinates using the **PerigeeSurface::localtoglobal()** method.

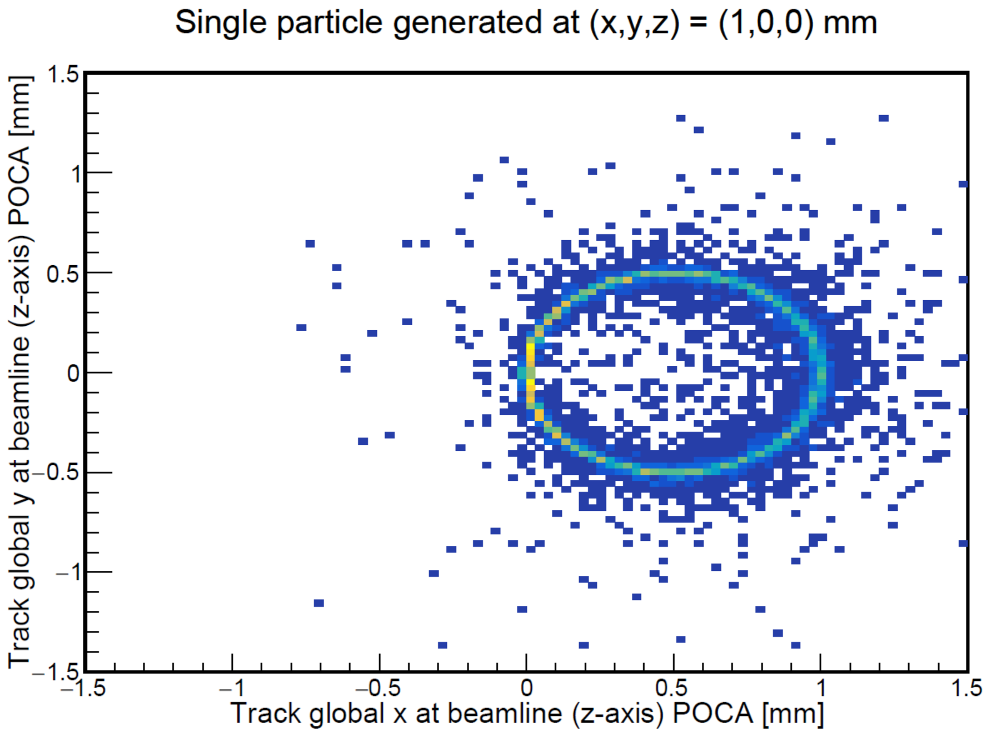
(vx, vy, vz) = (+1,0,0) mm



Reconstructed track parameters



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



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…

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/
```

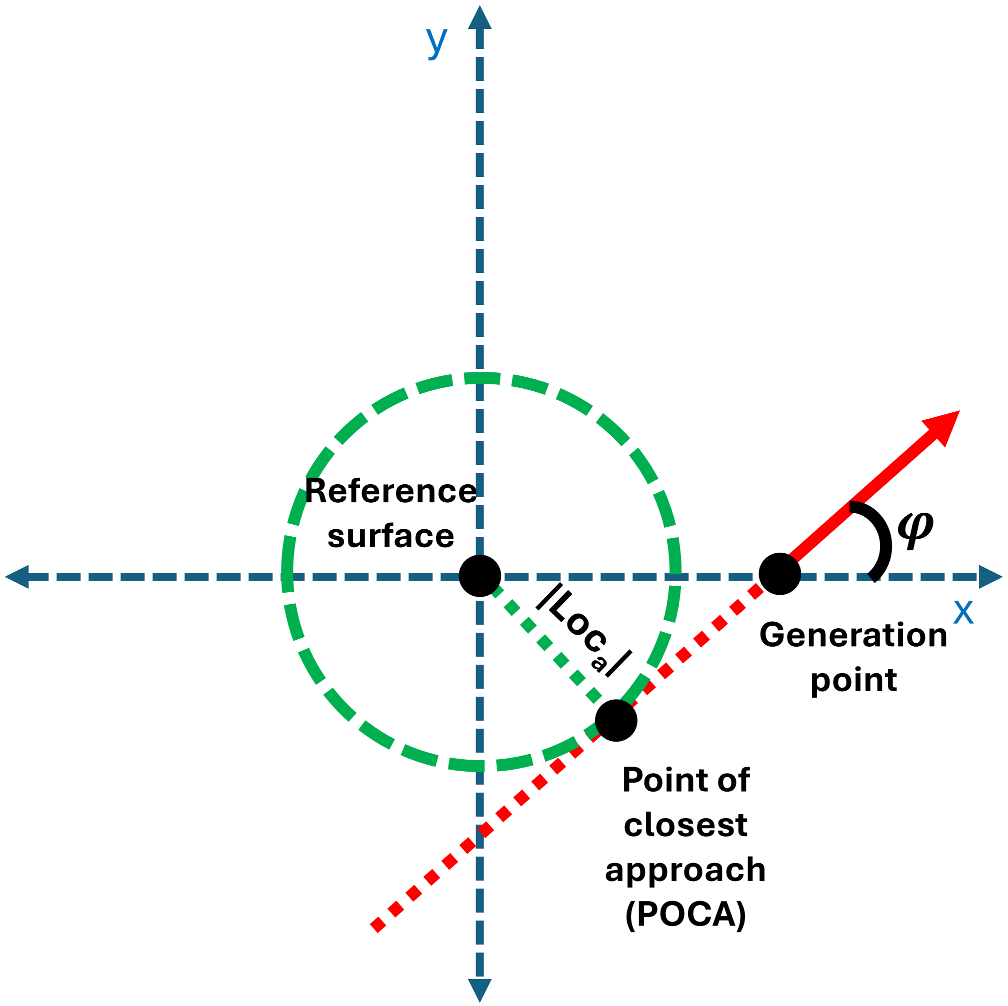
Running the analysis code

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
```

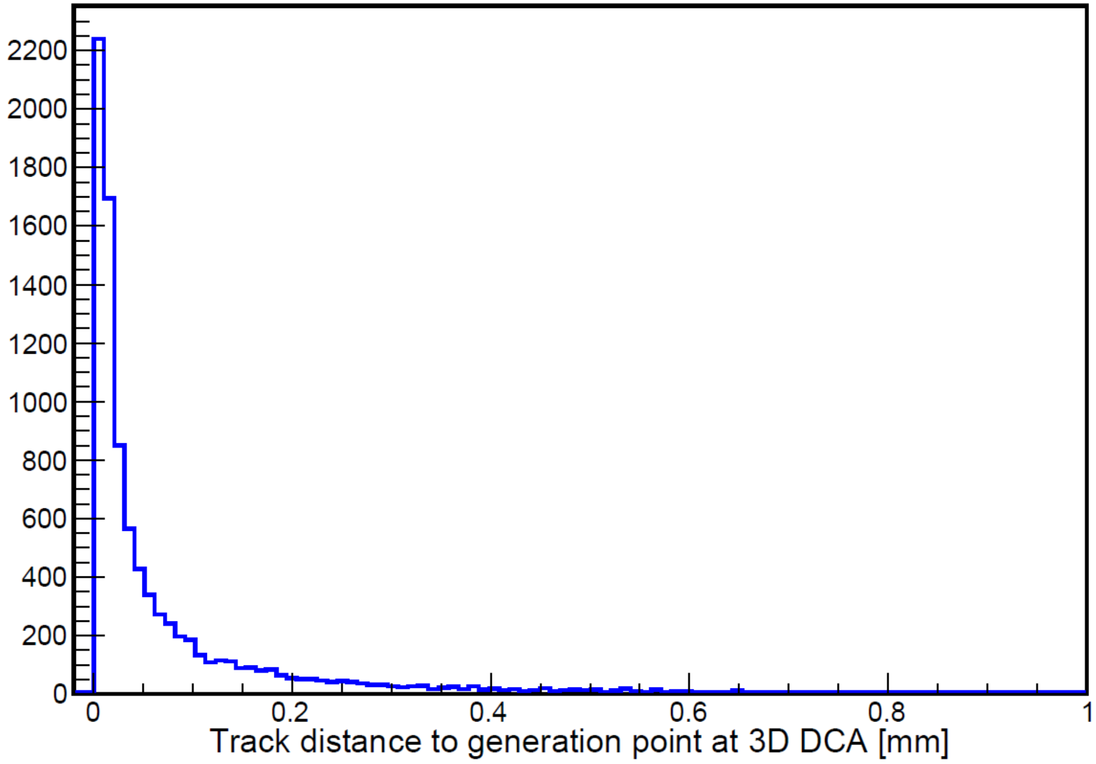
https://github.com/eic/snippets/tree/main/Tracking/ImpactPointEstimator

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

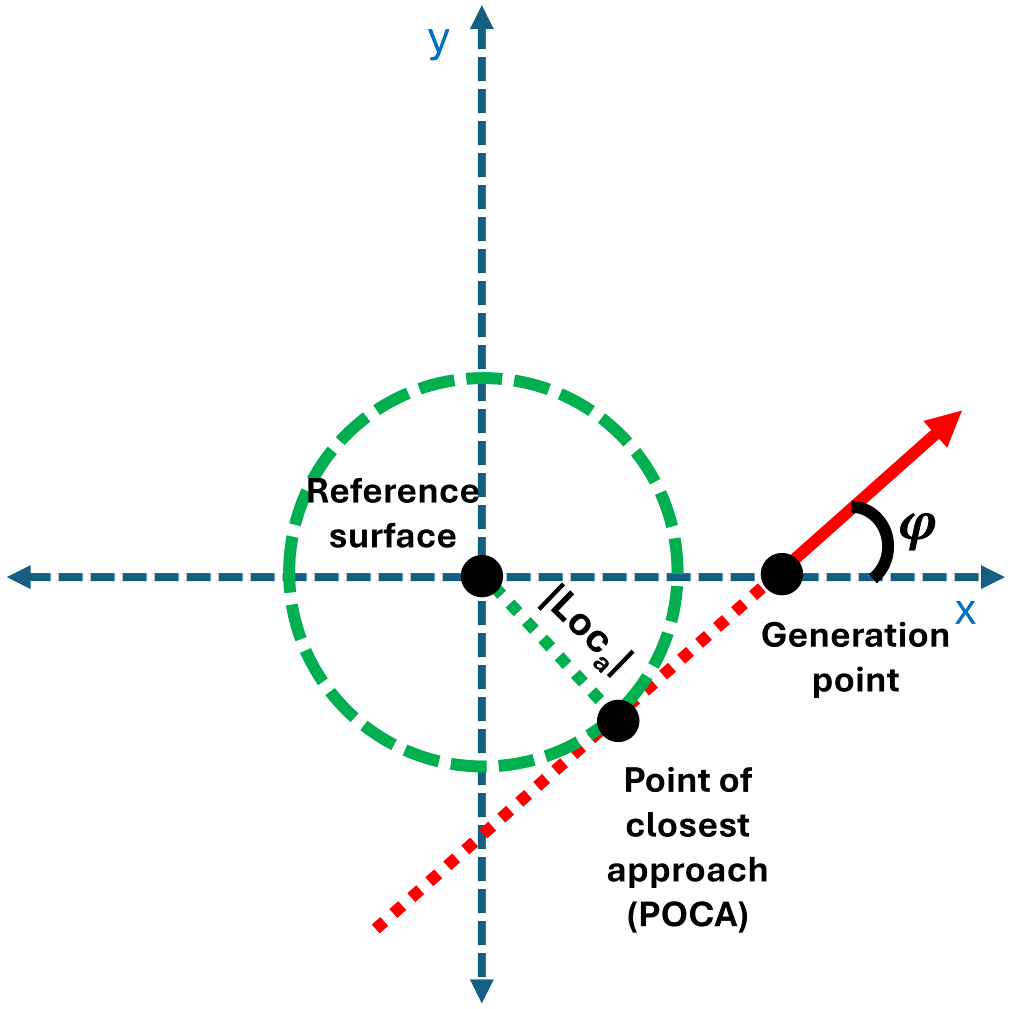


Since the particles are thrown from (vx, vy, vz) = (+1,0,0) mm, we expect this distance to be small.

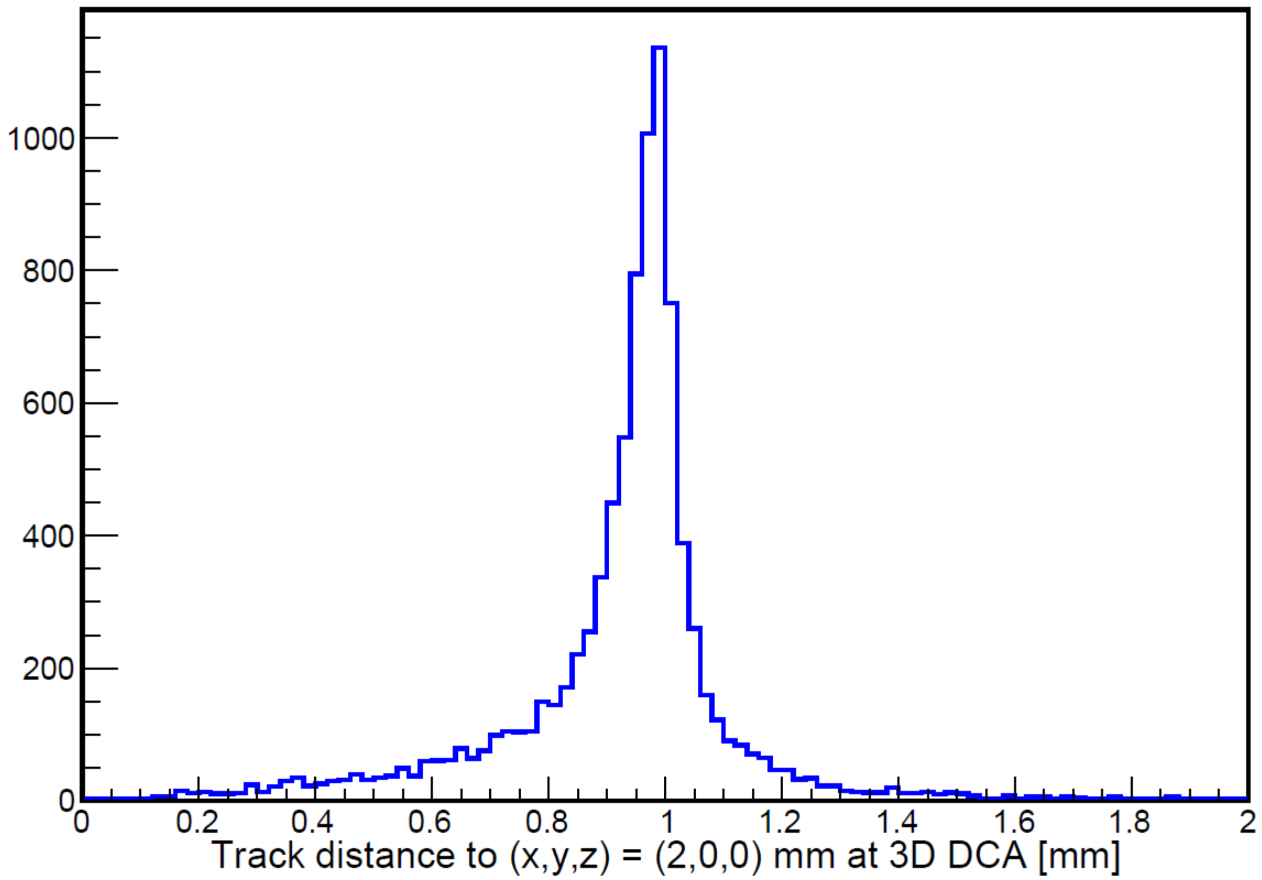
Single particle generated at (x,y,z) = (1,0,0) mm



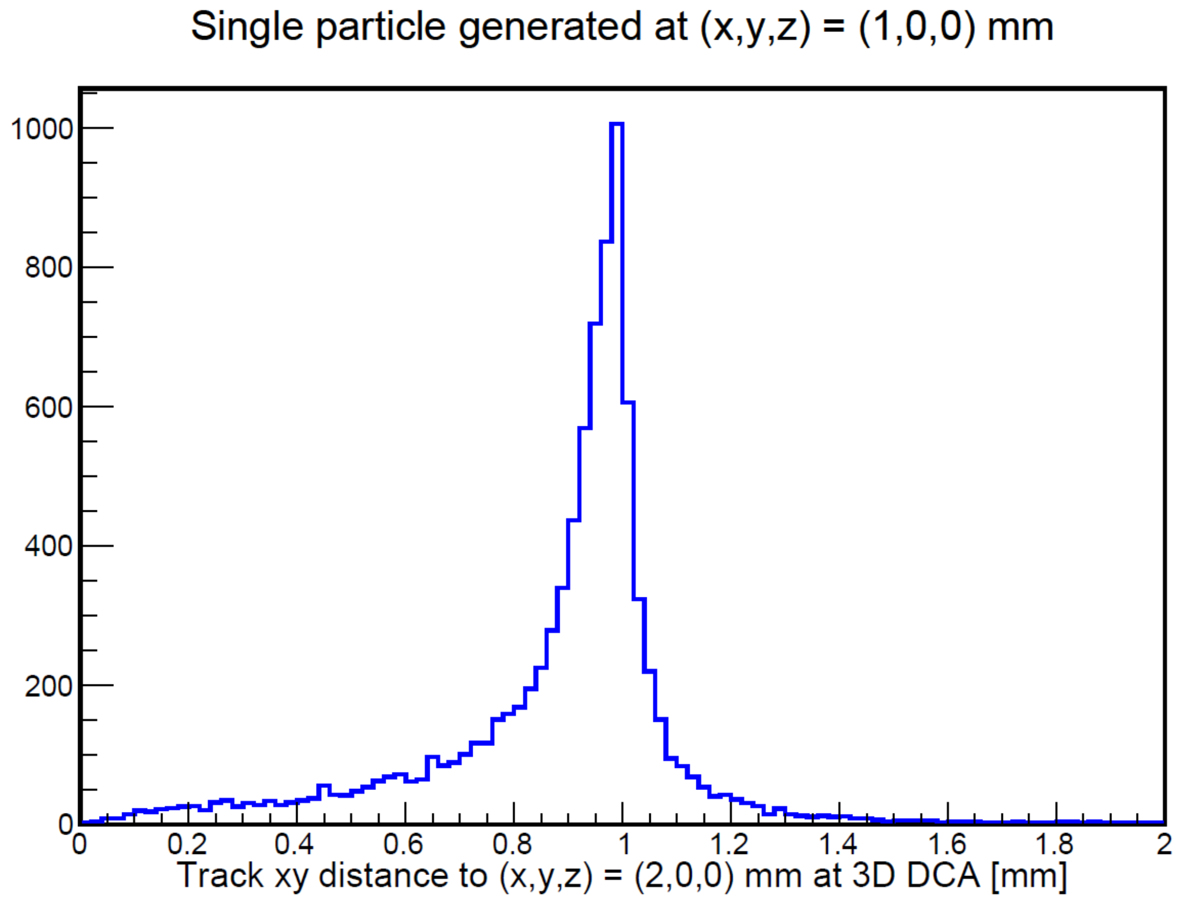
What if we calculate the closest distance to (x,y,z) = (2,0,0) mm?

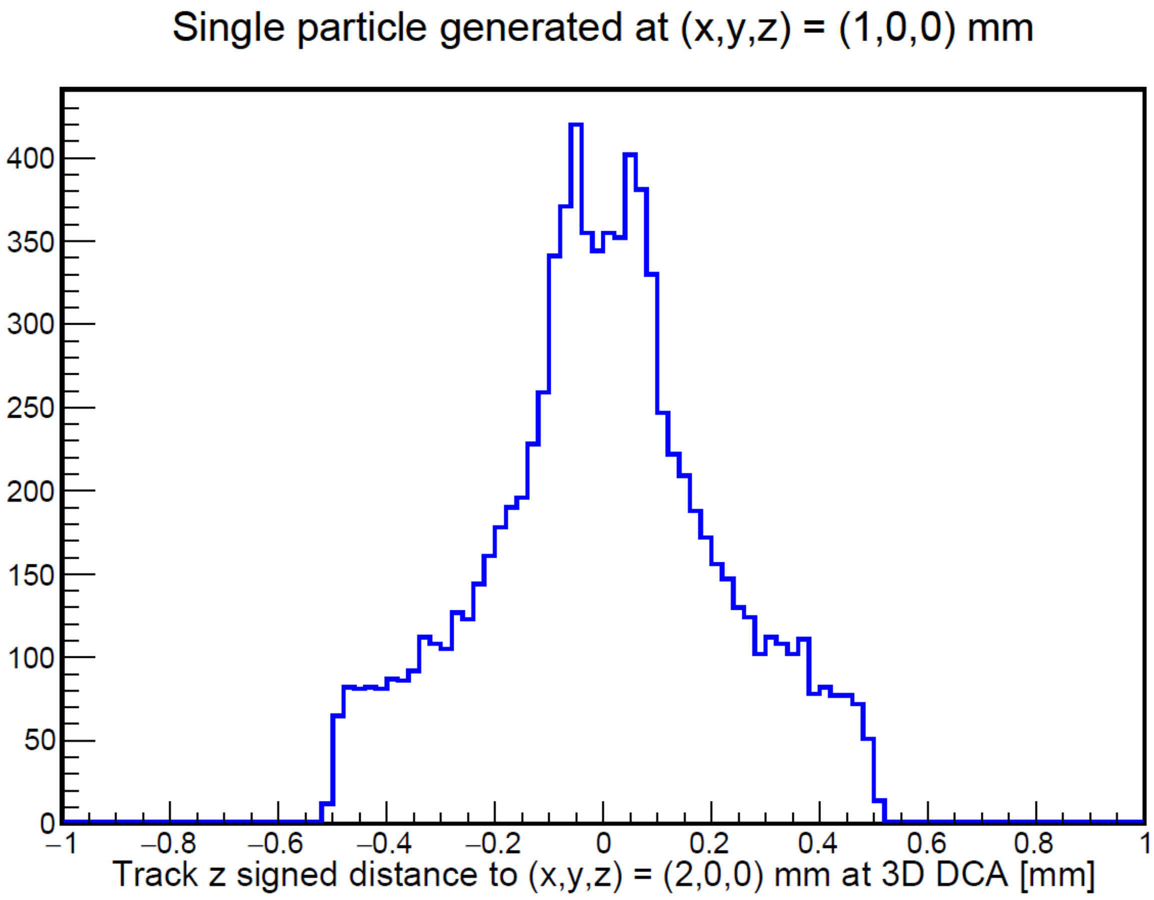


Single particle generated at (x,y,z) = (1,0,0) mm



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

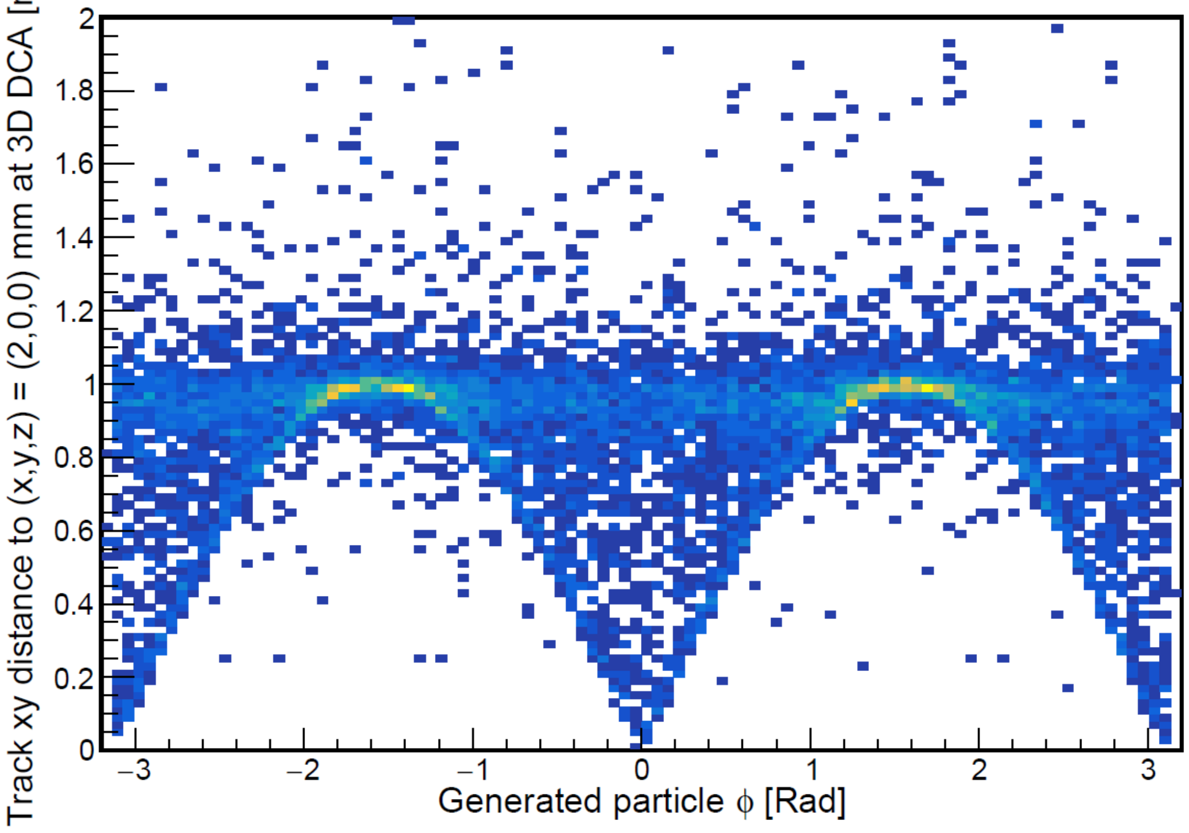




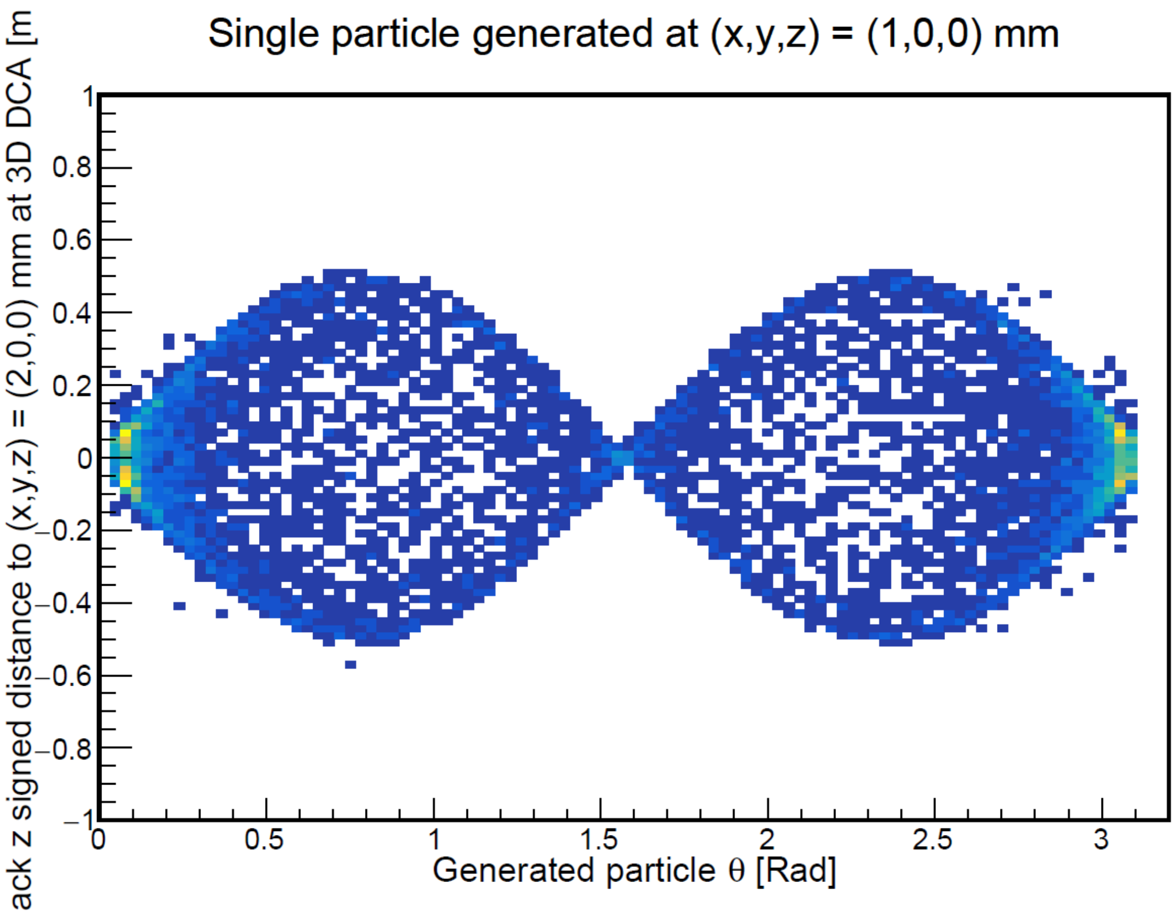
$$DCA_{xy} = \sqrt{(Trk_x - 2)^2 + (Trk_y - 0)^2} \text{ [mm]}$$

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

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

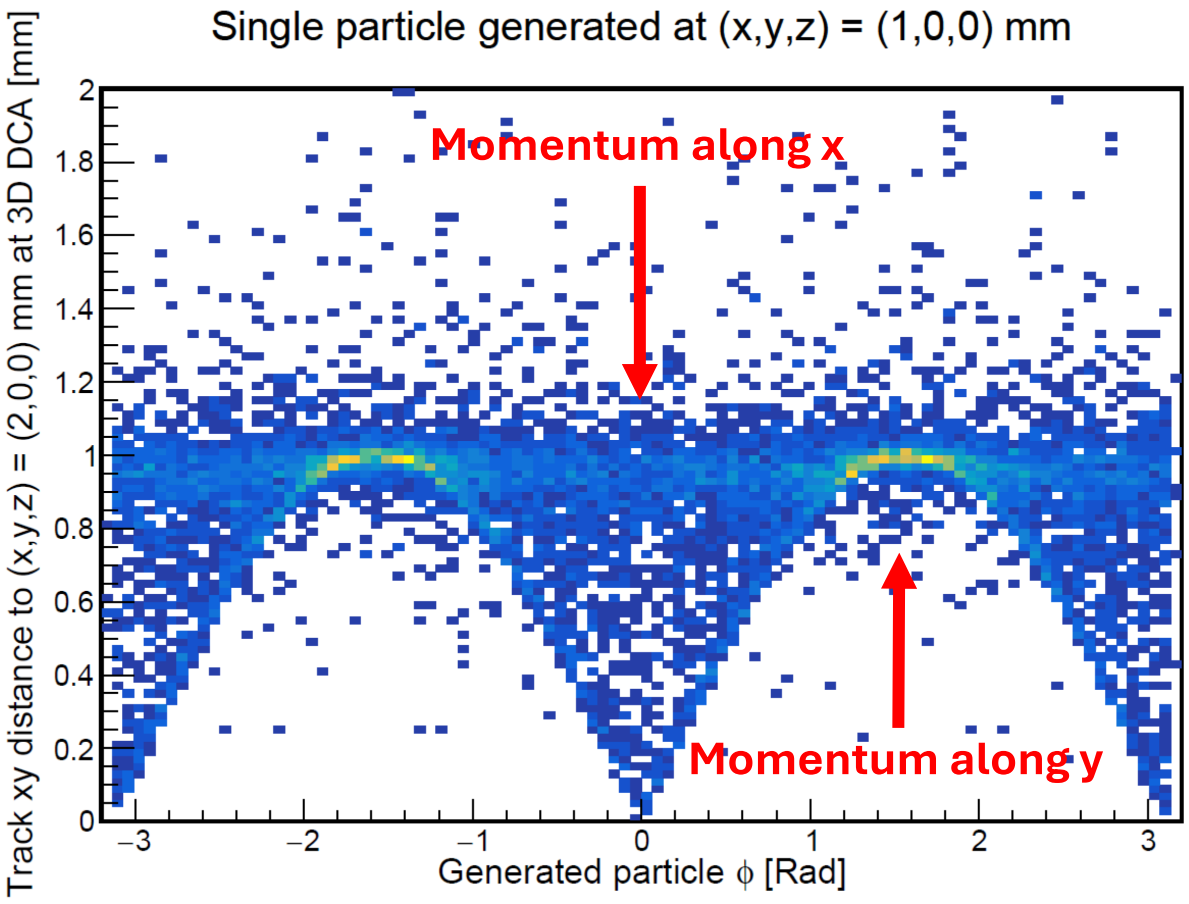


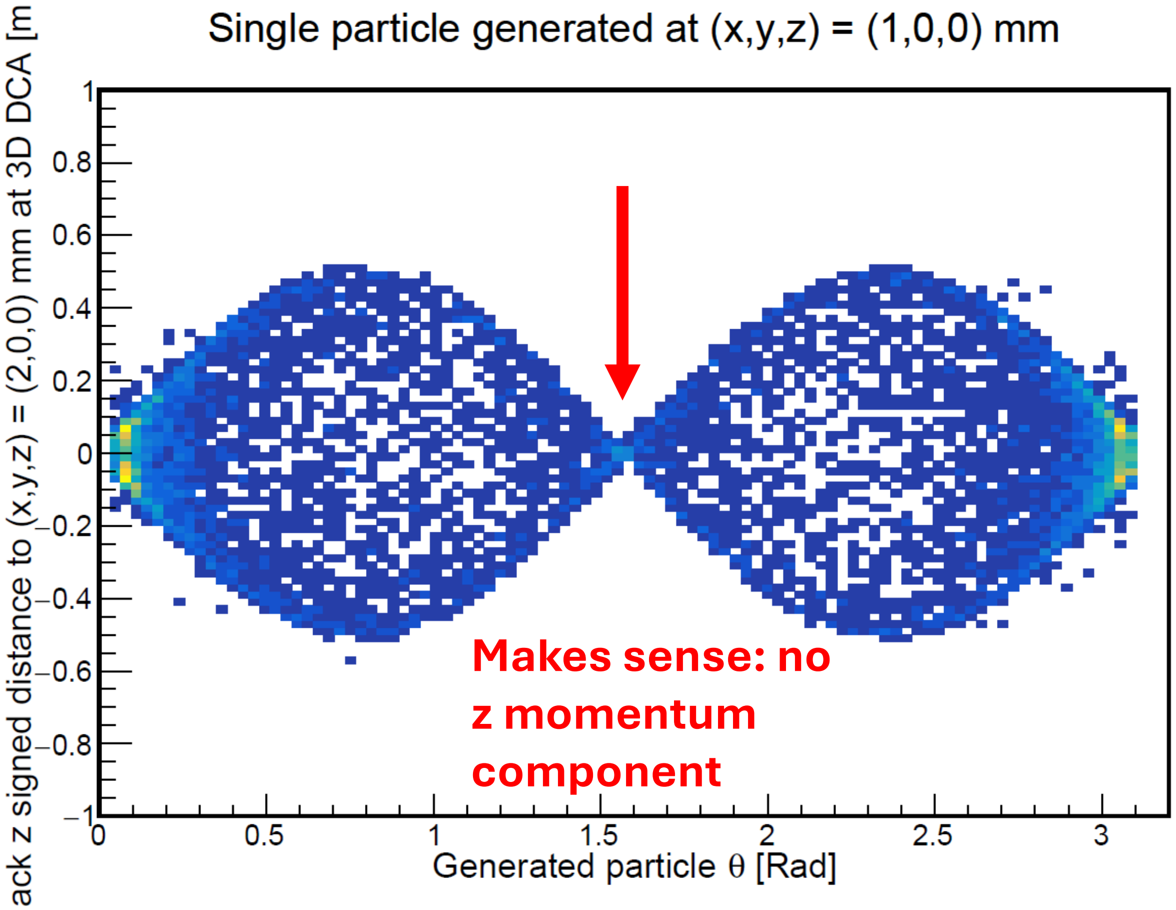
$$DCA_{xy} = \sqrt{(Trk_x - 2)^2 + (Trk_y - 0)^2}\ [\text{mm}]$$



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

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





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

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

Propagating track to Perigee surface at (x,y,z) = (2,0,0) mm

Propagates track to POCA in (x,y) plane to reference point.

```
//---- Part 3: Propagate track to Perigee surface at (x,y,z) = (2,0,0) mm ----

// Define Perigee surface to which to propagate track
auto perigee2 = Acts::Surface::makeShared<Acts::PerigeeSurface>(Acts::Vector3(2,0,0));

// Create propagator options
Acts::PropagatorOptions<> pOptions(trackingGeoCtx, fieldctx);
auto intersection = perigee2->intersect(trackingGeoCtx, track_parameters.position(trackingGeoCtx),
                    track_parameters.direction(), Acts::BoundaryCheck(false)).closest();

pOptions.direction = Acts::Direction::fromScalarZeroAsPositive(intersection.pathLength());

// Do the propagation to linPoint
auto result_perigee2 = propagator.propagateToSurface(track_parameters, *perigee2, pOptions);

if(result_perigee2.ok()){
    Acts::BoundTrackParameters trk_boundpar_perigee2 = result2.value();
    const auto& trk_perigee2  = trk_boundpar_perigee2.parameters();
 }
```

DCA between two tracks

- Some work has been done by Nicolas Schmidt on two-track vertexing using the Acts::FullBilloirVertexFitter algorithm: https://indico.bnl.gov/event/23598/contributions/92164/attachments/55516/94957/2024_06_SecondaryVertexing.pdf
- Some care needs to be taken about choosing the 'linearization' point. See chapter 5 in *CERN-THESIS-2010-027*.

More details in arXiv: 2102.08337